Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis: TD 5/98

An Abstract Interpretation Framework for Semantics and Diagnosis of Logic Programs

Marco Comini

Supervisor
G. Levi

Referee
R. Barbuti
G. Gallo
G. Ferrand
J. Maluszyński

March 1998

ADDR: Corso Italia 40, 56125 Pisa, Italy. TEL:+39-50-887111. FAX: +39-50-887226.
E-MAIL: comini@di.unipi.it. HOME-PAGE: http://www.di.unipi.it/~comini.

Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis: TD 5/98

An Abstract Interpretation Framework for Semantics and Diagnosis of Logic Programs

Marco Comini

Abstract. We define a semantic framework to reason about properties of abstractions of *SLD*-derivations (observables). The framework allows us to address problems such as the relation between the operational semantics and the denotational semantics and their properties (compositionality w.r.t. various syntactic operators, correctness and minimality, precision degree). Using abstract interpretation techniques to model abstraction allows us to state algebraic conditions on the observables which guarantee the validity of several compositionality and equivalence theorems.
We then use the semantic framework to extend declarative debugging techniques to cope with the analysis of operational properties, such as computed and correct answers, and of abstract properties, such as $depth(\mathsf{k})$ answers and groundness dependencies. The resulting technique (abstract diagnosis) leads to elegant bottom-up and top-down diagnosis methods, which do not require to determine the symptoms in advance, and which are effective in the case of abstract properties described by finite domains.

March 1998

ADDR: Corso Italia 40, 56125 Pisa, Italy. TEL:+39-50-887111. FAX: +39-50-887226.
E-MAIL: comini@di.unipi.it. HOME-PAGE: http://www.di.unipi.it/~comini.

To my parents, my sister, my friends and
my uncle Ezio, who died in 1987.
I still miss him.

Acknowledgments

First of all I would like to thank my parents for their unconditioned and unshakable trust in me. This helped and challenged me in doing more than I could not disappoint their expectations.

I thank my advisor, Giorgio Levi, for showing me the beauty of Logic Programming. He taught me the technique and the fairness not to take too seriously both me and my work.

I thank the people I worked with. Chiara Meo taught me to add rigor to my thoughts and not to take anything for granted. Giuliana Vitiello helped me to acquire self-confidence. Both of them also taught me how to interact with others.

I thank my colleagues of Pisa Computer Science Department. I got plenty of support from them. Special thanks to Roberto Giacobazzi, Maurizio Gabbrielli, Francesca Scozzari, Fausto Spoto, Alessandra Raffaetà, Enea Zaffanella and Roberto Bagnara for their friendship. Fausto and Francesca helped me in several ways in the last weeks, with their cooperation and availability. I would also like to thank professors Pierpaolo Degano and Andrea Maggiolo for their friendship.

Thanks to Jan Maluszyński and Ulf Nilsson for allowing me to finish my thesis with relative peace of mind. They also contributed to the final draft of this thesis, suggesting me how to write a good introduction.

My deep deep gratitude is devoted to my irreplaceable friends Bruno Rubino and Sarah Monaci who represent a cornerstone in my life. They did not let me get depressed, but they encouraged me to go on.

I thank all my special friends Domenico Tripepi, Alessandro Piombini, Massimo Rametta, Alberto Gelmini, Michele Pierini and Fausto Spoto for putting up with me during these last months. I know it has been hard.

I thank my friends Ignazio Guarrato, who helped me to improve my English, and Roberto Bonomo.

A special thank to my important old-date friends Paolo Cobelli and Pierivano Cherubini.

Most of all my greatest thanks go to my sister Elisabetta. Maybe she does not know yet that I am very proud of her.

Contents

Preface **vii**

Introduction **xi**
I.1 Computation Properties and Semantics . . . xii
I.2 Abstract Interpretation and Program Analysis . . . xiv
I.3 Program Debugging . . . xv
I.4 The Thesis Approach . . . xvii
I.5 Thesis Overview . . . xxi

1 Preliminaries **1**
1.1 Basic Set Theory . . . 1
1.1.1 Sets . . . 2
1.1.2 Relations and Functions . . . 3
1.2 Domain Theory . . . 6
1.2.1 Complete Lattices and Continuous Functions . . . 6
1.2.2 Fixpoint Theory . . . 9
1.3 Logic Programming . . . 10
1.3.1 Basic Logical Definitions . . . 10
1.3.2 Unification Theory . . . 12
1.3.3 Logic Programs and their Classical Semantics . . . 15
1.3.4 The s-semantics Approach . . . 24
1.4 Abstract Interpretation . . . 26
1.4.1 Closures on Complete Lattices . . . 26
1.4.2 Galois Insertions and Abstract Interpretation . . . 27
1.4.3 Correctness and Precision of Abstract Semantic Functions . . . 29
1.5 Quick Reference: Notation, Terminology and Basic Assumptions . . . 30

I A Theory of Observables for Logic Programs **35**

2 The Collecting Semantics of the Framework **37**

2.1 Semantic Domain 39
2.2 Denotational Semantics of *SLD*-derivations 41
2.2.1 Basic Operators on Derivations 44
2.2.2 Basic Operators on Collections 48
2.3 Operational Semantics 54
2.4 The Program Denotation 59
2.5 Semantic Properties of *SLD*-derivations 60
2.6 Discussions on the Results 71
2.7 Technical Proofs of the Chapter 72

3 The Abstraction Framework 89
3.1 The Observables 91
3.2 From the Observables to the Abstract Semantics 94
3.3 Perfect Observables 96
3.3.1 Computed Resultant Semantics 104
3.3.2 The Partial Proof Tree Semantics 106
3.4 Denotational Observables 107
3.4.1 The Computed Answer Observable and the s-semantics 109
3.4.2 The Correct Answer Observable 111
3.4.3 The Call Pattern Observable 112
3.5 Semi-denotational Observables 115
3.5.1 The Observable ψ for Groundness Analysis of Computed Answers 124
3.5.2 The *depth*(k) Observable 129
3.6 Semi-perfect Observables 130
3.6.1 *SLD*-trees with $\mathcal{POS}$ 142
3.7 Discussion on the Results of the Abstraction Framework 143
3.8 Applications and Future Developments 146
3.9 Proofs of Examples 147

II Abstract Diagnosis 163

4 Semantic Framework for Applications 165
4.1 The Concrete Collecting Semantics 168
4.1.1 Denotational Semantics 169
4.1.2 Operational Semantics 170
4.2 The Observables and the Abstract Semantics 173
4.2.1 Abstract Denotational Semantics 175
4.2.2 Abstract Operational Semantics 175

4.3 Complete Observables 176
4.3.1 The Correct Answer Observable 177
4.4 Approximate Observables 178
4.4.1 The $\mathcal{POS}$ Observable for Groundness Dependencies of Computed Answers 180
4.4.2 The $depth(\mathsf{k})$-answer Observable 182
4.5 Discussion on the Results of the Framework for Applications 183
4.6 Proofs of the Chapter 184

5 Abstract Diagnosis 197
5.1 Introduction 197
5.1.1 Declarative Debugging, Program Verification and Abstract Diagnosis 197
5.1.2 Program Properties and Abstract Interpretation 198
5.1.3 The Semantic Framework 198
5.1.4 Abstract Diagnosis 200
5.2 Abstract Diagnosis 202
5.2.1 Bottom-up Diagnosis 204
5.2.2 Top-down Diagnosis 205
5.2.3 The Diagnosis Meta-interpreters 206
5.2.4 Towards the Diagnosis Theorems 209
5.3 Abstract Diagnosis w.r.t. Complete Observables 210
5.3.1 Acceptable Programs 212
5.3.2 Discussion on Complete Observables 217
5.4 Partial Diagnosis 218
5.4.1 Discussion on Partial Diagnosis 223
5.5 Abstract Diagnosis w.r.t. Approximate Observables 224
5.5.1 Discussion on Approximate Observables 231
5.6 Modular Abstract Diagnosis 233
5.7 Discussion on the Results of Abstract Diagnosis 244
5.8 The Implementation of the Observable Modules 245
5.8.1 The Computed Answer Observable (Id) 245
5.8.2 The $depth(\mathsf{k})$ Observable (τ_{k}) 246
5.8.3 The $\mathcal{POS}$ Observable (Υ) 247

III Epilogue 255

Concluding Remarks 257

Bibliography **261**

Index **271**

List of Figures

2.1 The *SLD*-tree corresponding to the collection D in Example 2.1.1 . . 40
2.2 The *SLD*-trees corresponding to the collection D_1 in Example 2.1.1 . 41
2.3 The *SLD*-trees corresponding to the collection D_2 in Example 2.1.1 . 42
2.4 The append program 44
2.5 The $\mathcal{P}[\![\mathsf{P}]\!]$ operator of Example 2.2.1 45
2.6 The *SLD*-trees of Example 2.2.1 46
2.7 The *SLD*-tree corresponding to collection $\mathsf{q}(\mathsf{f}(\mathsf{g}(z)), z) \cdot \mathsf{D}_2$ in Example 2.2.5 50
2.8 The *SLD*-trees corresponding to collection $\mathsf{D}_1 \triangleright \mathsf{D}_2$ in Example 2.2.5 . 51
2.9 The *SLD*-trees corresponding to collection $\mathsf{D}_1 \times \mathsf{D}_2$ in Example 2.2.5 52
2.10 The ancestor program 55
2.11 The *SLD*-trees corresponding to collection D of Example 2.3.1 56
2.12 The *SLD*-tree transition $\mathsf{D}'' \underset{\mathsf{P}}{\longmapsto} \mathsf{D}''''$ of Example 2.3.1 57
2.13 The toy program of Example 2.4.1 60

3.1 The program of Example 3.4.5 111
3.2 The program P of Example 3.5.10 127
3.3 The program Q of Example 3.5.10 128

4.1 The sum program of Examples 4.1.1 and 5.5.4 171
4.2 The equational version of the ancestor program 172
4.3 The refutations of Example 4.1.3 173
4.4 The domain $\mathcal{POS}$ for two variables x, y 180
4.5 The program of Examples 4.4.3, 5.5.2 and 5.5.8 182
4.6 The program of Example 4.4.4 184

5.1 The program of Examples 5.2.4 and 5.2.7 204
5.2 The main module of the diagnosis programs 207
5.3 The common oracle interface of the observables modules 208
5.4 The wrong acceptable program of Example 5.3.12 215
5.5 The main module of the partial diagnosis program 221

5.6 The wrong reverse program of Example 5.4.11 222
5.7 The program of Example 5.5.3 226
5.8 The program of Example 5.5.9 231
5.9 The main module of the modular diagnosis program 235
5.10 The program of Example 5.6.8 237
5.11 The program of Example 5.6.14 242

Preface

What we are about to present is

- a unifying *framework* for standard and approximate semantics of logic programs based on abstract interpretation techniques; we will reconstruct some already existing semantics to prove the applicability of our ideas to some well known issues,

- *abstract diagnosis*, which is an application of the framework to program diagnosis; we illustrate its usefulness for debugging by using the various reconstructed semantics.

Motivations

Definite logic programs have a very elegant declarative semantics, i.e., the least Herbrand model. However, some semantics-based techniques (such as program analysis, debugging and transformation) require more traditional semantics which are able to capture computational rather than declarative properties. Several ad-hoc semantics modeling various abstraction of *SLD*-trees (observables) have been defined, including the ones specifically designed for static program analysis. The first motivation of this work was to develop a framework to *systematically* derive the semantics modeling an observable; to address problems such as the relation between the operational and the denotational semantics and to reason about their properties (e.g., compositionality, correctness and precision degree).

One interesting example of semantics-based technique (concerned with model-theoretic properties) which can take advantage of more concrete semantics is declarative debugging. Our second motivation was that of applying the results of the theoretical framework to extend declarative debugging techniques to cope with the analysis of abstract operational properties described by finite domains, such as groundness dependencies.

Contributions

The idea of using abstract interpretation techniques as unifying framework for various semantics is well-known [36]. However the originality of this thesis is that we fully exploit this idea and provide a real flexible framework which provides useful theoretical bases for *new* semantic-based applications.

The first contribution of this thesis is a uniform framework for the *reconstruction* of existing semantics and for the *systematic* design of *new* semantics able to deal also with the *approximation* typical of static *program analysis*. A framework where one can define denotations modeling various non-approximate observables (thus inheriting the basic constructions and results) was given in [53], by defining the observables by means of equivalence relations. More general semantic frameworks, which can also take into account approximation, can be defined using abstract interpretation. This is the approach taken in [20], where an observable is an abstraction according to abstract interpretation theory, and in [58], where abstract interpretation is used to discuss the relation among different semantics. [57] introduces a framework for non-standard semantics which are approximations of the computed answer semantics. Our framework encompasses the limitations of this ones. It deals with any abstraction of *SLD*-trees and uniformly accounts also approximation. Moreover we can systematically choose the appropriate level of abstraction for specializing the semantics operators of the framework.

The ingredients of our semantic framework are a concrete semantics (modeling *SLD*-trees) and an observable (abstraction of *SLD*-trees). The denotational semantics and the transition system for *SLD*-trees are defined in terms of four semantic operators, directly related to the syntactic structure of language. This allows us to reason about properties of the *SLD*-trees via an algebraic construction which gives new insights in the clarification of the proof space.

Using abstract interpretation techniques to model abstraction allows us to state algebraic conditions on several classes of observables w.r.t. the four basic semantic operators. These conditions guarantee the validity of several general theorems, once the correctness of the abstraction has been proven. Depending on the class, we automatically obtain a new denotational semantics, transition system, top-down and bottom-up denotations, together with several interesting theorems (equivalence, compositionality w.r.t. the various syntactic operators, correctness and minimality of the denotations and precision degree). Our taxonomy of observables is useful to design a *new* semantics with some *a priori* given properties. One needs only to define an observable and check if it

belongs to the right class, since the axioms for a class are sufficient conditions for the derived semantics to have the required properties.

The second contribution of this thesis is *abstract diagnosis*, which is an application of the framework to declarative debugging. Declarative debugging is concerned with model-theoretic properties. The related declarative semantics is the least Herbrand model in [91], the set of program completion models in [74] and the set of atomic logical consequences in [48]. The idea behind declarative debugging is to collect information about what the program is intended to do and compare this with what it actually does. By using these *symptoms*, a diagnoser can find errors.

The aim of abstract diagnosis is to extend declarative debugging to the case where the intended behavior (specification) of the diagnosed program is finite and define a program property rather than its semantics. The resulting technique shows that our approach is useful to define new elegant and powerful semantic-based techniques for programming tools. We propose some diagnosis techniques which would be of practical interest.

Partial diagnosis can be used whenever we have a (finite) partial knowledge about the intended behavior. This knowledge can be derived from symptom detection and symptom-directed queries to the user, as in the symptom-directed debuggers.

Diagnosis w.r.t. approximate observables is instead useful when one performs the diagnosis w.r.t. properties which can be modeled by abstractions over finite (Noetherian) domains. Finite specifications lead to the systematic derivation of the diagnosis algorithms from the underlying theory with *no need for symptom detection*.

Modular diagnosis shows that the diagnosis method does not need to be extended to perform the diagnosis in a modular way. We can verify and debug incomplete programs, once we have the specifications for the missing program components.

The theoretical results on partial correctness, completeness and bug derivation are valid for the *diagnosis algorithms* too. We show a prototype implementation of the algorithms by means of Prolog meta-programs.

Joint Work

Much of the content of this thesis has already been published. This dissertation is the compilation and adaptation of three major papers (which are about to appear

in journals) and several minor papers (which already appeared in international conference proceedings). Part I is based on previous work which I done together with G. Levi and M. C. Meo. Part II is based on previous work which I done together with G. Levi, M. C. Meo and G. Vitiello.

This work has evolved over a relatively long period of time. The framework originates from summer 1993 (by the ideas of my Master's Thesis) but has been revised and enriched a number of times. The work of abstract diagnosis dates back to autumn 1994, inspired by the framework results, and has evolved together with it. They both were subject to several revisions, the most recent as late as december 1997.

Organization

The thesis is divided into four parts. The first of these is introductory in nature. It consists in two chapters which provide goals and motivation for the work and preliminary material common to the following parts. The last part of the thesis contains a discussion about the results, conclusions and directions for future research. To help the reader we also introduced at the end an index of the notions and the symbols introduced in the thesis.

Each of the two intermediate parts is divided in two chapters. Each chapter begins with an introductory section intended to familiarize the reader with the area and to introduce specific conventions and terminology. Most of this material is not original work although much of it has been reformulated and adapted to harmonize with the rest of the text. Each chapter ends with a section discussing the problems addressed in the chapter, the impact of the work and its relations to previous work. We structured the text in such a way that both intermediate parts can be read independently of each other.

As already pointed out, the thesis is based on several papers with different notation and terminology. In compiling them we have tried to make the notation and terminology as coherent as possible. We have tried to the best of our ability to lift out material which is common to the parts and to place that material in Chapter 1. This chapter provides then the basic mathematical notation and the basic notions on semantics of logic programs and abstract interpretation.

Some examples of semantics are provided. It is not our goal to define new semantics, but mainly to reconstruct some already existing ones to prove the applicability of our ideas. However, is some cases this moves away from a mainly academic exercise, by introducing interesting theoretical problems. Technical proofs that do not contribute to the main results of the thesis are grouped into technical sections and are sometimes only outlined.

Introduction

A semantics for a programming language provides meanings for programs or, more generally, program components. The first objective of a semantics is to help understand the meaning of programs. However this is *not* the only one. Some successful applications are the semantics-based techniques (such as program analysis, debugging and transformation) which have been used to develop useful programming tools. One relevant example is the class of tools for debugging of logic programs.

Definite logic programs have a very elegant declarative semantics. The elegance of the classical approach to its semantics has long been considered one of the most appealing features of logic programming as well as the reason for its popularity. The classical semantics is a mathematical object which is defined in model-theoretic terms and which can be *computed* by a top-down construction (the success set) and by a bottom-up construction (the least fixpoint of the immediate consequence operator). However this approach to the semantics of logic languages has some limitations. One relevant missing feature is compositionality.

Lack of compositionality of conventional logic programming semantics has been a serious limitation, since by their very nature, logic program fragments are written to be used in an extensible, modular fashion. Moreover, even if the simplicity of the classical semantics of logic programming support different programming techniques, most of the existing semantics-based techniques require more concrete semantics which are able to capture computational rather than declarative properties. The difference between all these semantics is related to the *property of the computation* (observable) the semantics is intended to model. Some observables model declarative properties but many useful ones model operational properties. Since programs compute substitutions, reasoning about program behavior in terms of success sets (or equivalently in terms of Herbrand models) is inadequate.

This discussion should have convinced the reader that, even if the classical approach is appealing, it is not adequate to provide the semantic tools needed to develop real software which is based on the logic programming paradigm. What we need is a framework for defining semantics adequate to model any abstraction of *SLD*-trees (precise or approximated) and to reason about its properties. In partic-

ular, this framework should allow us to address problems such as

- the relation between the operational semantics and the denotational semantics
- the existence of a goal-independent denotation for a set of definite clauses
- the properties of the denotations (e.g., compositionality, correctness and minimality).

This is exactly the motivation of the work presented in this thesis. In addition, one of the main benefits of such a construction is to provide solid theoretical bases to support the development of new semantics-based techniques for programming tools or to extend some existing ones.

We will construct such a framework starting from a semantics modeling *SLD*-trees and then applying abstract interpretation techniques to define and classify observables. We will then use this framework to extend declarative debugging to cope with the analysis of operational properties, such as computed and correct answers, and of abstract properties, such as $depth(k)$ answers and groundness dependencies. The resulting technique (*abstract diagnosis*) shows that our approach is useful to define new elegant and powerful semantic-based techniques for programming tools.

The next sections introduce some terminology and deeper discussions concerning all these topics. Section I.4 describes our approach and Section I.5 give a brief account of the contents of the thesis.

I.1 Computation Properties and Semantics

Semantics can help to develop practical tools such as those which perform program analysis, program debugging and program transformation. For this purpose we seek notions of models which really capture the operational properties of logic programs and are, therefore, useful to define program equivalences and for semantics-based analysis.

A program can have *different* semantics depending on which properties of the computation (observables) we are interested in. A given choice of the observable α induces an *observational equivalence* on programs. Namely $P_1 \approx_\alpha P_2$ if and only if P_1 and P_2 are observationally indistinguishable according to α, or, in other terms, they have the same behavior w.r.t. the property which we are interested in. In pure logic programs we can be interested in different observables: if we are only concerned with the input-output behavior of programs, we should just observe computed answers and finite failures. However, there are tasks, such as program analysis and optimization, which force us to observe and take into account other features of the derivation. In principle one could be interested in the complete

information about the *SLD*-tree, namely the sequences of goals, most general unifiers and variants of clauses.

Defining an equivalence on programs $\approx_\alpha$ and a formal semantics $\mathcal{S}_\alpha[\![P]\!]$ are two strongly related tasks. If the semantics enjoys the property $P_1 \approx_\alpha P_2 \iff \mathcal{S}_\alpha[\![P_1]\!] = \mathcal{S}_\alpha[\![P_2]\!]$, then it identifies all and only the programs which are observationally indistinguishable w.r.t. α. Such an abstract model can be considered to be *the* semantics of a language w.r.t. a given observable: all the other semantics can be reduced to it by abstracting from the redundant information. Moreover this property is important, for instance, for deciding correctness of program transformation techniques.

For example if s denotes *successful derivations*, $P_1 \approx_s P_2$ if and only if, for any goal $\mathbf{G}$, $\mathbf{G}$ is refutable in P_1 if and only if $\mathbf{G}$ is refutable in P_2. This observable is adequate to characterize a theorem prover, yet it is definitely too abstract to capture the essence of logic programming, i.e., the ability to compute answers. The observable is the property which allows us to distinguish programs and is also the property we want to preserve in *equivalence preserving transformations*. Hence the most adequate observable in the case of logic programming is *computed answer substitutions* (denoted by ca), which are exactly the result of logic programs execution. $P_1 \approx_{ca} P_2$ if and only if, for any goal $\mathbf{G}$, $\mathbf{G}$ has the same computed answers in P_1 and in P_2.

As first shown in [45], the *van Emden and Kowalski semantics is not correct w.r.t. the observational equivalence based on computed answer substitutions*. Hence, when we have to analyze or transform programs, this semantics cannot be taken as reference semantics.

In addition to the problem related to modeling the computed answer substitution equivalences, there exists another problem with the least Herbrand model semantics. Namely a very important property, i.e., *compositionality* does not hold. Compositionality is considered one of the most desirable characteristics of a formal semantics, since it provides a foundation for program verification and modular design. Compositionality, also known as functoriality, has to do with a (syntactic) program construction operator $\circ$, and holds when the semantics of the compound construct $C_1 \circ C_2$ can be computed by composing the semantics of the constituents C_1 and C_2, i.e., if for a suitable homomorphism σ, $\mathcal{S}_\alpha[\![P_1 \circ P_2]\!] = \mathcal{S}_\alpha[\![P_1]\!]\, \sigma(\circ)\, \mathcal{S}_\alpha[\![P_2]\!]$.

Note that the typical operations that characterize a modular language are operations on the parts which compose the program rather than on the program itself. Separate compilation or analysis are examples of these operations. For instance we might want to apply certain optimizing transformations on a module, or to reason about the data flow within it, independently of the context that module belongs to. In the case of logic programs, the construct which raises a compositionality problem is the *union* of clauses (*OR*-compositionality). As we will see, the least Herbrand

model semantics of logic programs is not compositional w.r.t. the union. In fact the least Herbrand model of the union of two programs can not be determined by the least Herbrand model of the two programs.

I.2 Abstract Interpretation and Program Analysis

Abstract interpretation [32, 34], a technique for constructing verified analyses of program execution behavior, has been extensively applied to logic programming [65, 78, 57]. The relevant feature of abstract interpretation is that, once the property relevant for the analysis has been modeled by an abstract domain, we have a methodology to systematically derive an abstract semantics, which in turn allows us to effectively compute a (correct) approximation of the property. By using this approach, most of the theorem-proving, in the logical theory involved in program verification, boils down to computing on the abstract domain. This is obtained in general at the expense of precision.

A program analysis is viewed as a non-standard, abstract semantics defined over a domain of data description. An abstract semantics is constructed by replacing operations in a suitable concrete semantics with the corresponding abstract operations defined on data descriptions, namely, *abstract domains*. Such domains are called abstract because they abstract, from the concrete computation domain, the properties of interest.

The definition of an appropriate concrete semantics, capable of modeling those program properties of interest, is a key point in abstract interpretation [32] and semantic-based data-flow analysis. Program analyses are then defined by providing finitely computable abstract interpretations which preserve interesting aspects of program behavior. Formal justification of program analyses is reduced to proving conditions on the relation between data and data descriptions and on the elementary operations defined on the data description.

In program analysis, abstract interpretation theory is often used to establish the correctness of specific analysis algorithms and abstract domains. We are more concerned instead in its application to the systematic derivation of the (optimal) abstract semantics from the abstract domain. Recent results on domain refinement operators (see, for example, [50, 62, 89]) show that (optimal) abstract domains can systematically be derived from the property to be proved.

Abstract interpretation is inherently semantic sensitive and different semantic definition styles lead to different approaches to program analysis. In the case of logic programs, two main approaches exist, namely, the *top-down* and the *bottom-*

up approaches. The most popular approach is the top-down, which propagates the information as *SLD*-resolution does. In this class there are ad hoc algorithms, frameworks based on an operational semantics and frameworks based on a denotational semantics. The bottom-up approach propagates the information as in the computation of the least fixpoint of the immediate consequence operator T_P. The main difference between the top-down and the bottom-up approach is usually related to *goal dependency*. In particular, a top-down analysis starts with a specific goal, while the bottom-up approach determines an approximation of properties of programs which is goal independent. Note, however, that goal-independent (concrete and abstract) semantics can also be defined in a top-down way.

I.3 Program Debugging

The time and effort spent on validation of computer programs is known to take well over half of the total time for software development. Debugging, an essential ingredient in the testing stage of development, may also arise after the installation of the software and throughout its lifetime. Development of powerful programming languages, by easing the task of programming and resulting in higher level of complexity, can at the same time make debugging more tedious and difficult. Thus the emergence of automatic tools for debugging closely follows the development of new programming paradigms and languages.

Debugging of logic programs is a special case of the general problem of debugging. The role of debugging in general is to identify and eliminate differences between the intended semantics of a program and its actual semantics. We will assume that the user has a clear idea about the results that should be computed by the program. An error occurs when the program computes something that the programmer did not intend, or when it fails to compute something he wanted.

An advantage of logic programming languages is that they facilitate declarative programming. They make it possible, at least to a certain extent, to separate the declarative, logical semantics (what is computed) from the operational semantics (how it is computed). In principle, all the reasoning concerning the correctness of a program can be done on the level of the declarative semantics, thus abstracting from any details of the computation. This advantage is however lost when debugging has to be done on the level of the concrete computations. This happens in the case of usual debugging tools which are various versions of tracers. They force the programmer to think in terms of the sequences of computation steps. Thus it is important to develop debugging methods based on more abstract semantics which take into account *abstract* properties of the computation.

The debugging problem can formally be defined as follows. Let P be a program, $\mathcal{S}_\alpha[\![\mathsf{P}]\!]$ be the behavior of P w.r.t. the observable property α, and $\mathcal{I}_\alpha$ be the specification of the *intended* behavior of P w.r.t. α. Debugging consists in comparing $\mathcal{S}_\alpha[\![\mathsf{P}]\!]$ and $\mathcal{I}_\alpha$ and determining the wrong program components which are sources of errors, when $\mathcal{S}_\alpha[\![\mathsf{P}]\!] \neq \mathcal{I}_\alpha$. The formulation is parametric w.r.t. the property α considered in the specification $\mathcal{I}_\alpha$ and in the actual behavior $\mathcal{S}_\alpha[\![\mathsf{P}]\!]$.

Declarative debugging [91, 74, 48] is concerned with model-theoretic properties. The specification is the intended declarative semantics (the least Herbrand model in [91] and the set of atomic logical consequences in [48]). The idea behind declarative error diagnosis is to collect information about what the program is intended to do and compare this with what it actually does. By reasoning from this, a diagnoser can find errors. The information needed can be found by asking for example the user, a formal specification, or an older (correct) version of the program. The entities that provide the diagnoser with information are with a common term referred to as the *oracle*.

The declarative debugging method consists in two main techniques: incorrectness error diagnosis and insufficiency error diagnosis. Incorrectness symptoms are answers which are in the actual program semantics and are not in the intended semantics. The principal idea to find incorrectness errors is to inspect the *proof tree* constructed for an incorrectness symptom. To find the erroneous clause the diagnoser traverses the proof tree. At each node it asks the oracle about the validity of the corresponding atom. With the aid of the answers the diagnoser is able to identify the erroneous clause.

Insufficiency symptoms are answers which are in the intended semantics and are not in the actual program semantics. Hence insufficiency error diagnosis concerns the case when a program fails to compute some expected results. The objective for insufficiency diagnosis is to scrutinize the attempt to construct a proof for an atom which incorrectly fails. The reason for the error is located to one of the procedures (all clauses defining a predicate) in the computation.

The idea in declarative debugging to restrict attention to model-theoretic properties has some limitations.

- The most natural observable for the diagnosis of positive logic programs is computed answers. It leads to a more precise diagnosis technique than the declarative debugging in [91, 48], which can be reconstructed in terms of the more abstract observables *ground instances of computed answers* and *correct answers*.

- Debugging w.r.t. depth(k)-answers or groundness dependencies makes diagnosis effective, since both $\mathcal{I}_\alpha$ and $\mathcal{S}_\alpha[\![\mathsf{P}]\!]$ are finite.

- Debugging w.r.t. types allows us to detect bugs as the *inadmissible calls* in [85]. If $\mathcal{I}_\alpha$ specifies the intended program behavior w.r.t. types, abstract diagnosis boils down to type checking.

- Debugging w.r.t. modes and ground dependencies allows us to verify other partial program properties.

Thus, declarative debugging is not adequate to most real-life debugging problems and we need to extend it to cope with the analysis of operational properties and of abstract properties.

I.4 The Thesis Approach

This work is greatly influenced by the s-semantics approach [13] to the semantics of logic programs which has the ambition to model the operational behavior of a variety of logic languages. Here we will focus on pure logic programs. The aim of our approach is to model a given observable behavior (possibly in a compositional way) for pure logic languages. So we will have to consider any possible abstraction of *SLD*-trees. In the conclusions we will suggest some possible ways to move to other real-life logic languages.

The s-semantics approach is based on the idea of using denotations consisting of (equivalence classes of) *syntactic objects*. Because of the use of more syntax (like variables) in the semantic domains, they are more expressive than Herbrand interpretations. The idea is to provide a semantics of a program which enjoys several properties, for example the ability of the semantics to capture the observable behavior, so that two programs have the same denotation if and only if they are observationally equivalent (w.r.t. any goal $\mathbf{G}$).

Semantic definitions can be different in style, as in the case of the top-down *SLD*-resolution *operational semantics* and the bottom-up fixpoint *denotational semantics*. They can be different because of some of their properties. For example, *SLD*-resolution is *goal-dependent* since it allows to compute a denotation for a given goal. The fixpoint semantics is instead *goal-independent*, since it provides a denotation for a set of procedure declarations.

Some important properties of a semantics can be described as compositionality properties. One example is *OR*-compositionality, which tells us that the denotation of a set of clauses can be obtained by composing the denotations of the clauses themselves. Most of the existing goal-independent semantics, such as standard fixpoint semantics, are not *OR*-compositional. However, the most relevant difference is related to the *observable* the semantics is intended to model. Some observables

model declarative properties: an example is correct answer substitutions. However, most useful observables model operational properties: examples are resultants, proof trees, finite failures, computed answer substitutions, partial answers, call patterns, types and groundness dependencies.

In this thesis we push forward the s-semantics approach, by constructing a concrete semantics modeling *SLD*-trees and by combining it with the use of abstract interpretation techniques to model and classify *SLD*-tree abstractions (observables). This allow us to model, within the same framework, both the approximation which is involved in the abstractions used for program analysis and usual (precise) semantics. Using abstract interpretation techniques allows us to state algebraic conditions on the observables which guarantee the validity of several compositionality and equivalence theorems. This led to a flexible classification of the observables, where we can reason about properties such as *OR*-compositionality, existence of abstract transition system and precision degree. Actually, depending on the class, we automatically obtain

- a new denotational semantics,
- a transition system,
- top-down and bottom-up denotations,

together with several interesting theorems

- equivalence of operational and denotational semantics,
- compositionality w.r.t. the various syntactic operators,
- correctness and minimality of the denotations,
- precision degree.

The reconstruction of an *existing* semantics or the construction of a *new* semantics in the framework requires just a few very simple steps.

1. The formalization of the property we want to model as a Galois insertion $\langle \alpha, \gamma \rangle$ between *SLD*-derivations and the property domain.

2. The verification of some algebraic properties relating $\langle \alpha, \gamma \rangle$ and the basic semantic operators on *SLD*-derivations, to assign the property to the right class of observables.

3. Depending on the class, we automatically obtain the new denotational semantics, transition system, top-down and bottom-up denotations, together with several interesting theorems (equivalence, compositionality w.r.t. the various syntactic operators, correctness and minimality of the denotations).

We argue that the taxonomy is useful to design a new semantics with some *a priori* given properties. One need only to define an observable and check if it belongs to the right class, since the axioms for a class are sufficient conditions for the derived semantics to have the required properties. Furthermore, if the observable does not belong to the right class, by focusing on the axioms which are not satisfied, the "observable designer" can find useful suggestions on how to transform an unsatisfactory abstraction into a good observable, by using standard abstract interpretation techniques for the systematic construction of "more concrete" observables (e.g., reduced product, disjunctive completion, functional dependencies and Heyting completion).

The first class we consider is the one of *perfect observables*. We prove that perfect observables are precise and have all the properties of the concrete semantics.

For the class of *denotational observables*, we can obtain the optimal abstract semantics only in a denotational way, by taking the optimal abstract version of the semantic operator defining the denotation of the clauses. The abstract operational semantics is less precise. We prove that denotational observables have a precise abstract denotational semantics and that the abstract (goal-independent) denotation is correct, minimal and *AND*-compositional. Therefore, by moving from perfect to denotational observables, we lose the precision of the abstract transition system and *OR*-compositionality.

The third class of observables we study is the class of *semi-denotational observables*, intended to model some of the properties useful for static program analysis, where we give up precision to achieve termination in the construction of the abstract semantics. The semantics construction of semi-denotational observables is the same of denotational observables. We just lose the precision of the abstract denotational semantics (which is in any case more precise than the operational one). The denotation is the minimal *AND*-compositional denotation correct w.r.t. the observable.

The last class of observables we study is the class of *semi-perfect observables* which have all the properties of perfect observables apart from precision. In particular, they have equivalent operational and denotational semantics and their goal-independent denotations are both *AND*-compositional and *OR*-compositional. Let us just note that semi-perfect observables are essentially the observables which model top-down abstract interpretation frameworks (for example, [15, 69]).

As argued before, the aim of this semantics framework is to provide solid theoretical bases to support the development of new semantics-based techniques for programming tools or to extend some existing ones. To show our ideas at work, we considered *declarative debugging*. By applying our theoretical results, we showed how declarative debugging techniques can be extended to cope with the analysis of operational properties, such as computed and correct answers, and of abstract

properties, such as *depth*(k) answers and groundness dependencies. The resulting technique (*abstract diagnosis*) leads to elegant and powerful bottom-up and top-down diagnosis methods, which enjoy several interesting features when compared to the existing methods [91, 92, 48, 74].

- It is not required to determine the symptoms in advance.
- It allows us to obtain a more accurate diagnosis than the one that can be obtained using the least Herbrand model or the logical atomic consequence semantics.
- It is effective in the case of abstract properties described by finite (Noetherian) domains. The goal-independency property is essential to make the diagnosis finite.

In order for abstract diagnosis to be self-contained, we specialize the framework to practical application cases where we are only concerned with properties which are abstraction of the computed answer semantics and we are interested in equivalent top-down and bottom-up semantics. The denotational concrete semantics is essentially the s-semantics and the operational concrete semantics is obtained by further abstraction of *SLD*-trees (both given for an equational version of positive logic programs).

Only two classes of observables can arise (*complete* and *approximate*), depending on the precision degree. They enjoy several interesting semantic properties:

- there exists a goal-independent abstract top-down denotation (defined by means of a transition system), which is *AND*-compositional.
- there exists an abstract bottom-up denotation (defined by means of a denotational semantic definition), which is equivalent to the top-down one.
- the abstract transition system and the abstract denotational definition are systematically derived from the collecting semantics.

The difference between complete and approximate observables is related to precision. Namely, the abstract semantics coincides with the abstraction of the collecting semantics, in the case of complete observables, while it is just a correct approximation in the case of approximate observables.

Declarative debugging is based on the preliminary detection of *symptoms*, which are deviations from the expected behavior, which have to be determined by using other techniques, e.g., testing techniques. Our formalization of abstract diagnosis does not require to determine symptoms in advance. The systematic derivation of

all the errors in the program is possible because the semantics we use exhibit several useful properties (such as existence of a goal-independent top-down denotation equivalent to the fixpoint semantics, AND-compositionality, *etc*). The same properties allow us to relate naturally bottom-up and top-down diagnosis algorithms. Our diagnosis method is based on *incorrect clauses* and *uncovered elements*, which are detected by means of very simple bottom-up or top-down algorithms, without actually computing the abstract semantics (i.e., no fixpoint computation is required). In particular, we propose some diagnosis techniques which would be of practical interest.

Partial diagnosis can be used whenever we have a (finite) partial knowledge about the intended behavior. This knowledge can be derived from symptom detection (performed using testing techniques) and symptom-directed queries to the user, as in the symptom-directed debuggers. One might think of other partial knowledge acquisition techniques.

Diagnosis w.r.t. approximate observables is instead useful when one wants to perform the diagnosis w.r.t. properties which can be modeled by abstractions over Noetherian domains. This is interesting because several useful properties are modeled by approximate observables or by observables which can be systematically refined to approximate observables. Finite specifications lead to the systematic derivation of the diagnosis algorithms from the underlying theory with *no need for symptom detection*.

Modular diagnosis shows that the diagnosis method does not need to be extended to perform the diagnosis in a modular way. This is due to the fact that both the top-down and the bottom-up diagnosis algorithms are essentially based on the application of the "abstract immediate consequence operator" which is intrinsically compositional. This property shows that we can verify and debug incomplete programs, once we have the specifications for the missing program components.

I.5 Thesis Overview

In short, the thesis is organized as follows. Notations and basic concepts of logic programming (syntax and semantics) are introduced in Chapter 1. After having shown the classical results, we give also an introduction to abstract interpretation.

In Chapter 2 we define a semantic framework to reason about compositional properties of *SLD*-trees and their abstractions. We obtain a kind of 'collecting' semantics, which gives the maximum amount of information on computations and allows us to

observe all the internal details of *SLD*-trees. Moreover, we address problems such as relation between the operational semantics and the denotational semantics, the existence of a denotation for a set of definite clauses and their properties (compositionality w.r.t. various syntactic operators, correctness and minimality). The denotational semantics and the transition system for *SLD*-trees are defined in terms of four semantic operators, directly related to the syntactic structure of language. The proofs of all main theorems heavily rely on some basic results, which express properties of the *concrete* primitive semantic operators.

In Chapter 3 we define the semantic framework together with a flexible classification of the observables, where we can reason about properties such as *OR*-compositionality and existence of an abstract transition system. The observables are formalized by using Galois insertions between the domain of the *SLD*-trees and abstract domains. Using abstract interpretation techniques to model abstraction allows us to state very simple conditions on several classes of observables which guarantee the validity of several general theorems, once the correctness of the abstraction has been proved. By using the interaction between the properties of the *abstraction* and the properties of the *concrete* primitive semantic operators, we can easily inherit in the abstract case all those properties of the collecting semantics for which the suitable lemmata on the semantic operators hold. The reconstruction of several existing semantics is shown.

In Chapter 4 we specialize the framework to practical application cases where we are only concerned with properties which are abstraction of the computed answer semantics and we are interested in equivalent top-down and bottom-up semantics. This framework is deeply used in Chapter 5 to show how declarative debugging techniques can be extended to cope with the analysis of operational properties (such as computed and correct answers) and of abstract properties (such as $depth(\mathsf{k})$ answers and groundness dependencies). The resulting technique (abstract diagnosis) leads to elegant bottom-up and top-down diagnosis methods, which do not require to determine the symptoms in advance, and which are effective in the case of abstract properties described by finite domains.

Finally, the last chapter is devoted to some conclusive remarks.

Chapter 1

Preliminaries

Most of the notations used in this thesis are introduced in this chapter. The experienced readers are advised to have a look at the "notation summary" Section 1.5 first.

Essentially, this chapter presents the informal, logical and set-theoretic notations and concepts we will use to write down and reason about our ideas. The chapter is simply presented by using an informal extension of our everyday language to talk about mathematical objects like sets; it is not to be confused with the formal definitions about them that we will encounter later.

Some more specific notions will be introduced in the chapters where they are needed. For the terminology not explicitly shown and for a more motivated introduction about fixpoint theory and algebraic notation, the reader can consult [77, 11, 10]. We will refer to [75, 3] for further details on the logic programming theory.

1.1 Basic Set Theory

To define the basic notions we will use the standard (meta) logical notation to denote conjunction, disjunction, quantification and so on (*and*, *or*, *for each*, ...). We will use some informal logical notation in order to stop our mathematical statements getting out of hand. For statements (or assertions) A and B, we will commonly use abbreviations like:

A, B	for (A and B), the conjunction of A and B,
$\mathsf{A} \Longrightarrow \mathsf{B}$	for (A implies B), which means (if A then B),
$\mathsf{A} \Longleftrightarrow \mathsf{B}$	for (A if and only if B), which expresses the logical equivalence of A and B.

We will also make statements by forming disjunctions (A or B), with the self-evident meaning, and negations (not A), sometimes written $\neg A$, which is true if and only if A is false. It is a tradition to write $x \not< y$ instead of $\neg(x < y)$.

A statement like $P(x, y)$, which involves variables x, y, is called a predicate (or property, or relation, or condition) and it only becomes true or false when the pair x, y stands for particular things. We use logical quantifiers $\exists$ (read "there exists") and $\forall$ (read "for all") to write assertions like $\exists x.\, P(x)$ as abbreviating "for some x, $P(x)$" or "there exists x such that $P(x)$", and $\forall x.\, P(x)$ as abbreviating "for all x, $P(x)$" or "for any x, $P(x)$". The statement $\exists x, y, \ldots, z.\, P(x, y, \ldots, z)$ abbreviates $\exists x. \exists y. \cdots \exists z.\, P(x, y, \ldots, z)$, and $\forall x, y, \ldots, z.\, P(x, y, \ldots, z)$ abbreviates $\forall x. \forall y. \cdots \forall z.\, P(x, y, \ldots, z)$. In order to specify a set S over which a quantifier ranges, we write $\forall x \in S.\, P(x)$ instead of $\forall x.\, x \in S \implies P(x)$, and $\exists x \in S.\, P(x)$ instead of $\exists x.\, x \in S, P(x)$.

1.1.1 Sets

Intuitively, a set is an (unordered) collection of objects, which are elements (or members) of it. We write $a \in S$ when a is an element of the set S. Moreover, we write $\{a, b, c, \ldots\}$ for the set of elements $a, b, c, \ldots$.

A set S is said to be a subset of a set S', written $S \subseteq S'$, if and only if every element of S is an element of S', i.e., $S \subseteq S' \iff \forall z \in S.\, z \in S'$. A set is determined solely by its elements in the sense that two sets are equal if and only if they have the same elements. So, sets S and S' are equal, written $S = S'$, if and only if every element of S is an element of S' and vice versa.

Sets and Properties

A set can be determined by a property P. We write $S := \{x \mid P(x)\}$, meaning that the set S has as elements precisely all those x for which $P(x)$ is true. We will not be formal about it, but we will avoid trouble like Russell's paradox and will have at the same time a world of sets rich enough to support most mathematics. This will be achieved by assuming that certain given sets exist right from the start and by using safe methods for constructing new sets.

We write $\emptyset$ for the null or empty set and $\mathbb{N}$ for the set of natural numbers $0, 1, 2, \ldots$.

The cardinality of a set S is denoted by $card(S)$. A set S is called *denumerable* if $card(S) = card(\mathbb{N})$ and *countable* if $card(S) \leq card(\mathbb{N})$.

Constructions on Sets

Let S be a set and $P(x)$ be a property. By $\{x \in S \mid P(x)\}$ we denote the set $\{x \mid x \in S, P(x)\}$. Sometimes, we will use a further abbreviation. Suppose $E(x_1, \ldots, x_n)$ is some expression which for particular elements $x_1 \in S_1, \ldots, x_n \in S_n$ yields a particular element and $P(x_1, \ldots, x_n)$ is a property of such $x_1, \ldots, x_n$. We use $\{E(x_1, \ldots, x_n) \mid x_1 \in S_1, \ldots, x_n \in S_n, P(x_1, \ldots, x_n)\}$ to abbreviate $\{y \mid \exists x_1 \in S_1, \ldots, x_n \in S_n . y = E(x_1, \ldots, x_n), P(x_1, \ldots, x_n)\}$.

The *powerset* of a set S, $\{S' \mid S' \subseteq S\}$, is denoted by $\wp(S)$.

Let I be a set. By $\{x_i\}_{i \in I}$ (or $\{x_i \mid i \in I\}$) we denote the set of (unique) objects x_i, for any $i \in I$. The elements x_i are said to be *indexed* by the elements $i \in I$.

The *union* of two sets is $S \cup S' := \{a \mid a \in S \text{ or } a \in S'\}$. Let $\mathcal{S}$ be a set of sets, $\bigcup \mathcal{S} = \{a \mid \exists S \in \mathcal{S} . a \in S\}$. When $\mathcal{S} = \{S_i\}_{i \in I}$, for some indexing set I, we write $\bigcup \mathcal{S}$ as $\bigcup_{i \in I} S_i$. The *intersection* of two sets is $S \cap S' := \{a \mid a \in S, a \in S'\}$. Let $\mathcal{S}$ be a nonempty set of sets. Then $\bigcap \mathcal{S} := \{a \mid \forall S \in \mathcal{S} . a \in S\}$. When $\mathcal{S} = \{S_i\}_{i \in I}$ we write $\bigcap \mathcal{S}$ as $\bigcap_{i \in I} S_i$.

The *cartesian product* of S and S' is the set $S \times S' := \{(a, b) \mid a \in S, b \in S'\}$, the set of ordered pairs of elements with the first from S and the second from S'. More generally $S_1 \times S_2 \times \cdots \times S_n$ consists of the set of n-tuples $(x_1, \ldots, x_n)$ with $x_i \in S_i$ and S^n denotes the set of n-tuples of elements in S.

$S \setminus S'$ denotes the set where all the elements from S, which are also in S', have been removed, i.e., $S \setminus S' := \{x \mid x \in S, x \notin S'\}$.

1.1.2 Relations and Functions

A *binary relation* between S and S' ($R : S \times S'$) is an element of $\wp(S \times S')$. We write $x \mathrel{R} y$ for $(x, y) \in R$.

A *partial function* from S to S' is a relation $f : S \times S'$ for which $\forall x, y, y' . (x, y) \in f, (x, y') \in f \Longrightarrow y = y'$. By $f : S \rightharpoonup S'$ we denote a partial function of the set S (the *domain*) into the set S' (the *range*). The set of all partial functions from S to S' is denoted by $[S \rightharpoonup S']$. Moreover, we use the notation $f(x) = y$ when there is a y such that $(x, y) \in f$ and we say $f(x)$ is defined, otherwise $f(x)$ is undefined. Sometimes, when $f(x)$ is undefined, we write $f(x) = \aleph$, where $\aleph$ denotes the undefined element. For each set S we assume that $\aleph \subseteq S$, $\aleph \cup S = S$ and $\emptyset \not\subseteq \aleph$. This will be formally motivated in Section 1.2.1.

Given a partial function $f : S \rightharpoonup S'$, the sets $\mathit{supp}(f) := \{x \in S \mid f(x) \text{ is defined}\}$ and $\mathit{img}(f) := \{f(x) \in S' \mid \exists x \in S . f(x) \text{ is defined}\}$ are, respectively, the *support* and the *image* of f. A partial function is said to be *finite-support* if $\mathit{supp}(f)$ is finite. Moreover, it is said to be *finite* if both $\mathit{supp}(f)$ and $\mathit{img}(f)$ are finite. In the following, we will often use finite-support partial functions. Hence, to simplify the

notation, by

$$f := \begin{cases} v_1 \mapsto r_1 \\ \quad \vdots \\ v_n \mapsto r_n \end{cases}$$

we will denote (by cases) any function f which assumes on input values $v_1, \ldots, v_n$ output values $r_1, \ldots, r_n$ and is otherwise undefined. Furthermore, if the support of f is just the singleton $\{v\}$, we will denote it by $f := v \mapsto r$.

A (total) function f from S to S' is a partial function from S to S' such that, for all $x \in S$, there is some $y \in S'$ such that $f(x) = y$. That is equivalent as saying that f is total if $\mathit{supp}(f) = S$. Although total functions are a special kind of partial function, it is a tradition to understand something described as simply a function to be a total function. So we will always say explicitly when a function is partial. To indicate that a function f from S to S' is total, we write $f : S \to S'$. Moreover, the set of all (total) functions from S to S' is denoted by $[S \to S']$.

A function $f : S \to S'$ is *injective* if and only if for each $x, y \in S$ if $f(x) = f(y)$ then $x = y$. f is *surjective* if and only if for each $x' \in S'$ there exists $x \in S$ such that $f(x) = x'$.

We denote by $f = g$ the extensional equality, i.e., for each $x \in S$, $f(x) = g(x)$. Furthermore, $g := f[{}^{v}/_{x}]$ denotes the function g which differs from f only for the assignment of v to x, i.e., $g(x) = v$ and, for each $y \neq x$, $g(y) = f(y)$.

Lambda Notation

It is sometimes useful to use the lambda notation to describe functions. It provides a way of referring to functions without having to name them. Suppose $f : S \to S'$ is a function which, for any element $x \in S$, gives a value $f(x)$ which is exactly described by expression E, probably involving x. Then we can write $\lambda x \in S.\, E$ for the function f. Thus, $(\lambda x \in S.\, E) := \{(x, E[x]) \mid x \in S\}$ and so $\lambda x \in S.\, E$ is just an abbreviation for the set of input-output values determined by the expression $E[x]$. We use the lambda notation also to denote partial functions by allowing expressions in lambda-terms that are not always defined. Hence, a lambda expression $\lambda x \in S.\, E$ denotes a partial function $S \rightharpoonup S'$ which, on input $x \in S$, assumes the value $E[x] \in S'$, if the expression $E[x]$ is defined, and otherwise it is undefined.

Composing Relations and Functions

We compose relations, and so partial and total functions, $R : S \times S'$ and $Q : S' \times S''$ by defining their *composition* (a relation between S and S'') by $Q \circ R := \{(x, z) \in S \times$

$S'' \mid y \in S', (x, y) \in R, (y, z) \in Q\}$. R^n is the relation

$$\underbrace{R \circ \cdots \circ R}_{n},$$

i.e., $R^1 := R$ and (assuming R^n is defined) $R^{n+1} := R \circ R^n$. Each set S is associated with an identity function $Id_S := \{(x, x) \mid x \in S\}$, which is the neutral element of $\circ$. Thus we define $R^0 := Id_S$.

The *transitive and reflexive closure* R^* of a relation R on S is $R^* := \bigcup_{i \in \mathbb{N}} R^i$.

The *function composition* of $g : S \rightharpoonup S'$ and $f : S' \rightharpoonup S''$ is the partial function $f \circ g : S \rightharpoonup S''$, where $(f \circ g)(x) := f(g(x))$, if $g(x)$ (first) and $f(g(x))$ (then) are defined, and it is otherwise undefined. When it is clear from the context $\circ$ will be omitted.

A function $f : S \to S'$ is *bijective* if it has an *inverse* $g : S' \to S$, i.e., if and only if there exists a function g such that $g \circ f = Id_S$ and $f \circ g = Id_{S'}$. Then the sets S and S' are said to be in 1-1 correspondence. Any set in 1-1 correspondence with a subset of natural numbers $\mathbb{N}$ is said to be countable. Note that a function f is bijective if and only if it is injective and surjective.

Direct and Inverse Image of a Relation

We extend relations, and thus partial and total functions, $R : S \times S'$ to functions on subsets by taking $R(X) := \{y \in S' \mid \exists x \in X. (x, y) \in R\}$ for $X \subseteq S$. The set $R(X)$ is called the *direct image* of X under R. We define $R^{-1}(Y) := \{x \in S \mid \exists y \in Y. (x, y) \in R\}$ for $Y \subseteq S'$. The set $R^{-1}(Y)$ is called the *inverse image* of Y under R. Thus, if $f : S \rightharpoonup S'$ is a partial function, $X \subseteq S$ and $X' \subseteq S'$, we denote by $f(X)$ the *image* of X under f, i.e., $f(X) := \{f(x) \mid x \in X\}$ and by $f^{-1}(X')$ the *inverse image* of X' under f, i.e., $f^{-1}(X') := \{x \mid f(x) \in X'\}$.

Equivalence Relations and Congruences

An *equivalence relation* $\approx$ on a set S is a binary relation on S ($\approx : S \times S$) such that, for each $x, y, z \in S$,

$x \mathrel{R} x$	(reflexivity)
$x \mathrel{R} y \Longrightarrow y \mathrel{R} x$	(symmetricity)
$x \mathrel{R} y, y \mathrel{R} z \Longrightarrow x \mathrel{R} z$	(transitivity)

The *equivalence class* of an element $x \in S$, with respect to $\approx$, is the subset $[x]_\approx := \{y \mid x \approx y\}$. When clear from the context we abbreviate $[x]_\approx$ by $[x]$ and often abuse notation by letting the elements of a set denote their correspondent equivalence

classes. The *quotient set* $S/_{\approx}$ of S modulo $\approx$ is the set of equivalence classes of elements in S (w.r.t. $\approx$).

An equivalence relation $\approx$ on S is a *congruence* w.r.t. a partial function $f : S^n \rightharpoonup S$ if and only if, for each pair of elements $a_i,\ b_i \in S$ such that $a_i \approx b_i$, (if $f(a_1, \ldots, a_n)$ is defined then also $f(b_1, \ldots, b_n)$ is defined and)

$$f(a_1, \ldots, a_n) \approx f(b_1, \ldots, b_n).$$

Then, we can define the partial function $f_{\approx} : (S/_{\approx})^n \rightharpoonup S/_{\approx}$ as

$$f_{\approx}([a_1]_{\approx}, \ldots, [a_n]_{\approx}) := [f(a_1, \ldots, a_n)]_{\approx},$$

since, given $[a_1]_{\approx}, \ldots, [a_n]_{\approx}$, the class $[f(a_1, \ldots, a_n)]_{\approx}$ is uniquely determined independently of the choice of the representatives $a_1, \ldots, a_n$.

1.2 Domain Theory

We will present here the (abstract) concepts of complete lattices, continuous functions and fixpoint theory, which are the standard tools of denotational semantics.

1.2.1 Complete Lattices and Continuous Functions

A binary relation $\leq$ on S ($\leq : S \times S$) is a *partial order* if, for each $x, y \in S$,

$$x \leq x \qquad \text{(reflexivity)}$$
$$x \leq y, y \leq x \Longrightarrow x = y \qquad \text{(antisymmetry)}$$
$$x \leq y, y \leq z \Longrightarrow x \leq z \qquad \text{(transitivity)}$$

A *partially ordered set* (poset) $(S, \leq)$ is a set S equipped with a partial order $\leq$. A set S is *totally ordered* if it is partially ordered and, for each $x, y \in S$, $x \leq y$ or $y \leq x$. A *chain* is a (possibly empty) totally ordered subset of S.

A *preorder* is a binary relation which is reflexive and transitive. A preorder $\leq$ on a set S induces on S an equivalence relation $\approx$ defined as follows: for each $x, y \in S$,

$$x \approx y \Longleftrightarrow x \leq y, y \leq x.$$

Moreover, $\leq$ induces on $S/_{\approx}$ the partial order $\leq_{\approx}$ such that, for each $[x]_{\approx}, [y]_{\approx} \in S/_{\approx}$,

$$[x]_{\approx} \leq_{\approx} [y]_{\approx} \Longleftrightarrow x \leq y.$$

A binary relation $<$ is *strict* if and only if it is anti-reflexive (i.e., not $x < x$) and transitive.

Given a poset $(S, \leq)$ and $X \subseteq S$, $y \in S$ is an *upper bound* for X if and only if, for each $x \in X$, $x \leq y$. Moreover, $y \in S$ is the *least upper bound* (called also join) of X, if y is an upper bound of X and, for every upper bound y' of X, $y \leq y'$. A least upper bound of X is often denoted by $lub_S\, X$ or by $\bigsqcup_S X$. We also write $\bigsqcup_S\{d_1, \ldots, d_n\}$ as $d_1 \sqcup_S \cdots \sqcup_S d_n$. Dually an element $y \in S$ is a *lower bound* for X if and only if, for each $x \in X$, $y \leq x$. Moreover, $y \in S$ is the *greatest lower bound* (called also meet) of X, if y is a lower bound of X and for every lower bound y' of X, $y' \leq y$. A greatest lower bound of X is often denoted by $glb_S\, X$ or by $\bigsqcap_S X$. We also write $\bigsqcap_S\{d_1, \ldots, d_n\}$ as $d_1 \sqcap_S \cdots \sqcap_S d_n$. When it is clear from the context, the subscript S will be omitted. Moreover $\bigsqcup\{D_i\}_{i\in I}$ and $\bigsqcap\{D_i\}_{i\in I}$ can be denoted by $\bigsqcup_{i\in I} D_i$ and $\bigsqcap_{i\in I} D_i$. It is easy to check that if *lub* and *glb* exist, then they are unique.

Complete Partial Orders and Lattices

A *direct set* is a poset in which any subset of two elements (and hence any finite subset) has an upper bound in the set. A *complete partial order* (*CPO*) S is a poset such that every chain D has the least upper bound (i.e., there exists $\bigsqcup D$). Notice that any set ordered by the identity relation forms a *CPO*, of course without a bottom element. Such *CPO*s are called discrete. We can add a bottom element to any poset $(S, \leq)$ which does not have one (even to a poset which already has one). The new poset $S_\perp$ is obtained by adding a new element $\perp$ to S and by extending the ordering $\leq$ as $\forall x \in S.\, \perp \leq x$. If S is a discrete *CPO*, then $S_\perp$ is a *CPO* with bottom element, which is called flat.

A *complete lattice* is a poset $(S, \leq)$ such that for every subset X of S there exists $\bigsqcup X$ and $\bigsqcap X$. Let $\top$ denote the *top element* $\bigsqcup S = \bigsqcap \emptyset$ and $\perp$ denote the *bottom element* $\bigsqcap S = \bigsqcup \emptyset$ of S. The elements of a complete lattice are thought of as points of information and the ordering as an approximation relation between them. Thus, $x \leq y$ means x approximates y (or, x has less or the same information as y) and so $\perp$ is the point of least information. It is easy to check that, for any set S, $\wp(S)$ under the subset ordering $\subseteq$ is a complete lattice, where $\sqcup$ is union, $\sqcap$ is intersection, the top element is S and the bottom element is $\emptyset$. Also $(\wp(S))_\perp$ is a complete lattice.

Given a complete lattice $(L, \leq)$, the set of all partial functions $F = [S \rightharpoonup L]$ inherits the complete lattice structure of L. Let simply define $f \preceq g := \forall x \in S.\, f(x) \leq g(x)$, $(f \sqcup g)(x) := f(x) \sqcup g(x)$, $(f \sqcap g)(x) := f(x) \sqcap g(x)$, $\perp_F := \lambda x \in S.\, \perp_L$ and $\top_F := \lambda x \in S.\, \top_L$. In the following $f + g$ will be used to denote $f \sqcup g$.

Continuous and Additive Functions

Let $(\mathsf{L}, \leq)$ and $(\mathsf{M}, \sqsubseteq)$ be (complete) lattices. A function $\mathsf{f} : \mathsf{L} \to \mathsf{M}$ is *monotonic* if and only if

$$\forall \mathsf{x}, \mathsf{y} \in \mathsf{L}.\, \mathsf{x} \leq \mathsf{y} \Longrightarrow \mathsf{f}(\mathsf{x}) \sqsubseteq \mathsf{f}(\mathsf{y}).$$

Moreover, f is *continuous* if and only if, for each non-empty chain $\mathsf{D} \subseteq \mathsf{L}$,

$$\mathsf{f}(\bigsqcup_{\mathsf{L}} \mathsf{D}) = \bigsqcup_{\mathsf{M}} \mathsf{f}(\mathsf{D}).$$

Every continuous function is also monotonic, since $\mathsf{x} \leq \mathsf{y} \Longrightarrow \bigsqcup_{\mathsf{M}}\{\mathsf{f}(\mathsf{x}), \mathsf{f}(\mathsf{y})\} = \mathsf{f}(\bigsqcup_{\mathsf{L}}\{\mathsf{x}, \mathsf{y}\}) = \mathsf{f}(\mathsf{y}) \Longrightarrow \mathsf{f}(\mathsf{x}) \sqsubseteq \mathsf{f}(\mathsf{y})$.

Complete partial orders correspond to types of data (data that can be used as input or output to a computation) and computable functions are modeled as continuous functions between them.

A partial function $\mathsf{f} : \mathsf{S} \rightharpoonup \mathsf{S}'$ is *additive* if and only if the previous continuity condition is satisfied for each non-empty set. Hence, every additive function is also continuous. Dually we define *co-continuity* and *co-additivity*, by using $\sqcap$ instead of $\sqcup$.

It can be proved that the composition of monotonic, continuous or additive functions is, respectively, monotonic, continuous or additive.

The mathematical way of expressing that structures are "essentially the same" is through the concept of isomorphism which establishes when structures are isomorphic. A continuous function $\mathsf{f} : \mathsf{D} \to \mathsf{E}$ between *CPO*s D and E is said to be an *isomorphism* if there is a continuous function $\mathsf{g} : \mathsf{E} \to \mathsf{D}$ such that $\mathsf{g} \circ \mathsf{f} = Id_{\mathsf{D}}$ and $\mathsf{f} \circ \mathsf{g} = Id_{\mathsf{E}}$. Thus f and g are mutual inverses. This is actually an instance of a general definition which applies to a class of objects and functions between them (*CPO*s and continuous functions in this case). It follows from the definition that isomorphic *CPO*s are essentially the same but for a renaming of elements. It can be proved that a function $\mathsf{f} : \mathsf{D} \to \mathsf{E}$ is an isomorphism if and only if f is bijective and, for all $\mathsf{x}, \mathsf{y} \in \mathsf{D}$, $\mathsf{x} \leq_{\mathsf{D}} \mathsf{y} \Longleftrightarrow \mathsf{f}(\mathsf{x}) \leq_{\mathsf{E}} \mathsf{f}(\mathsf{y})$.

Function Space

Let D, E be *CPO*s. It is a very important fact that the set of all continuous functions from D to E can be made into a complete partial order. The function space $[\mathsf{D} \to \mathsf{E}]$ consists of continuous functions $\mathsf{f} : \mathsf{D} \to \mathsf{E}$ ordered pointwise by $\mathsf{f} \sqsubseteq \mathsf{g} \Longleftrightarrow \forall \mathsf{d} \in \mathsf{D}.\, \mathsf{f}(\mathsf{d}) \sqsubseteq \mathsf{g}(\mathsf{d})$. This makes the function space a complete partial order. Note that, provided E has a bottom element $\bot_{\mathsf{E}}$, such a function space of *CPO*s has a bottom element, the constantly $\bot_{\mathsf{E}}$ function $\bot_{[\mathsf{D} \to \mathsf{E}]} := \lambda \mathsf{d} \in \mathsf{D}.\, \bot_{\mathsf{E}}$. Least upper bounds of

chains of functions are given pointwise, i.e., a chain of functions $f_0 \sqsubseteq f_1 \sqsubseteq \ldots \sqsubseteq f_n \sqsubseteq \ldots$ has *lub* $\bigsqcup_{[D \to E]} f_n := \lambda d \in D.\ \bigsqcup_E \{f_n(d)\}_{n \in \mathbb{N}}$.

It is not hard to see that the partial functions $L \rightharpoonup D$ are in 1-1 correspondence with the (total) functions $L \to D_\perp$, and that, in this case, any total function is continuous; the inclusion order between partial functions corresponds to the "pointwise order" $f \sqsubseteq g \iff \forall \sigma \in L.\ f(\sigma) \sqsubseteq g(\sigma)$ between functions $L \to D_\perp$. Because partial functions from a *CPO* so does the set of functions $[L \to D_\perp]$ ordered pointwise. This is the reason why we assumed that, for each set S, $\aleph \subseteq S$, $\aleph \cup S = S$ and $\emptyset \not\subseteq \aleph$.

1.2.2 Fixpoint Theory

Given a poset $(S, \leq)$ and a function $f : S \to S$, a *fixpoint* of f is an element $x \in S$ such that $f(x) = x$. A *pre-fixpoint* of f is an element $x \in S$ such that $f(x) \leq x$ and dually a *post-fixpoint* of f is an element $x \in S$ such that $x \leq f(x)$. Moreover, we say that $x \in S$ is the *least fixpoint of* f (denoted by *lfp* f) if and only if x is a fixpoint of f and for all fixpoints y of f, $x \leq y$. Dually, we define the *greatest fixpoint* (denoted by *gfp* f).

The fundamental theorem of Knaster-Tarski states that the set of fixpoints of a monotonic function f is a complete lattice.

Theorem 1.2.1 (Fixpoint theorem) *[96] A monotonic function f on a complete lattice $(L, \leq)$ has the least fixpoint and the greatest fixpoint. Moreover,*

$$\mathit{lfp}(f) = \bigsqcap\{x \mid f(x) \leq x\} = \bigsqcap\{x \mid x = f(x)\}$$
$$\mathit{gfp}(f) = \bigsqcup\{x \mid x \leq f(x)\} = \bigsqcup\{x \mid x = f(x)\}.$$

The Knaster-Tarski Theorem is important because it applies to any monotone function on a complete lattice. However, most of the time we will be concerned with least fixpoints of continuous functions which we will construct by the techniques of the previous section, as least upper bounds of chains in a *CPO*. Therefore, it's useful to state some more notations and results on fixpoints of continuous functions defined on (complete) lattices.

First of all we have to introduce the notion of *ordinal*. We assume that an ordinal is a set, where every element of an ordinal is still an ordinal and the class of ordinals is ordered by membership relation ($\alpha < \beta$ means $\alpha \in \beta$). Consequently, every ordinal coincides with the set of all smaller ordinals. The least ordinals are 0, $1 := \{0\}$, $2 := \{0, \{0\}\}$, *etc.* Intuitively, the class of ordinals is the transfinite sequence $0 < 1 < 2 < \ldots < \omega < \omega+1 < \ldots < \omega+\omega < \ldots < \omega^\omega$, *etc.* Ordinals will be often denoted by Greek letters. An ordinal γ is a *limit ordinal* if it is neither 0 nor the successor of an ordinal; so, if $\beta < \gamma$, then there exists σ such that $\beta < \sigma < \gamma$. The

first limit ordinal, which is equipotent with the set of natural numbers, is denoted (by an abuse of notation) by ω. Often, in the definitions of *CPO* and of continuity, directed sets are used instead of chains. It is possible to show that if the set S is denumerable, then the definitions are equivalent.

The ordinal powers of a monotonic function $T : S \to S$ on a *CPO* S are defined as

$$T\uparrow_\alpha(x) := \begin{cases} x & \text{if } \alpha = 0 \\ T(T\uparrow_{(\alpha-1)}(x)) & \text{if } \alpha \text{ is a successor ordinal} \\ \bigsqcup\{T\uparrow_\beta(x) \mid \beta < \alpha\} & \text{if } \alpha \text{ is a limit ordinal.} \end{cases}$$

In the following, we will use the standard notation $T\uparrow_\alpha := T\uparrow_\alpha(\bot)$, where $\bot$ is the least element of S. In particular, $T\uparrow_\omega := \bigsqcup_{n<\omega} T\uparrow_n$, $T\uparrow_{n+1} := T(T\uparrow_n)$, for $n < \omega$, and $T\uparrow_0 := \bot$, where $\bigsqcup$ is the *lub* operation of S. Sometimes, $T\uparrow_\alpha(x)$ may be denoted simply by $T^\alpha(x)$.

The next important result is usually attributed to Kleene and gives an explicit construction of the least fixpoint of a continuous function f on a *CPO* D.

Theorem 1.2.2 (Fixpoint Theorem) *Let* $f : D \to D$ *be a continuous function on a CPO* D *and* $d \in D$ *be a pre-fixpoint of* f*. Then* $\bigsqcup\{f\uparrow_n(d) \mid n \leq \omega\}$ *is the least fixpoint of* f *greater than* d*. In particular* $f\uparrow_\omega$ *is the least pre-fixpoint and least fixpoint of* f*.*

Each *CPO* D with bottom $\bot$ is associated with a fixpoint operator *fix* : $[D \to D] \to D$, *fix* $:= \bigsqcup_{n<\omega}(\lambda f.\, f^n(\bot))$, i.e., *fix* is the least upper bound of the chain of the functions $\lambda f.\, \bot \sqsubseteq \lambda f.\, f(\bot) \sqsubseteq \lambda f.\, f(f(\bot)) \sqsubseteq \ldots$, where each of these is continuous and so an element of the *CPO* $[[D \to D] \to D]$.

1.3 Logic Programming

In this section we will introduce some notations about logic programming that will be used in the following and, for the reader's convenience, we will recall some basic notions. We refer to [75, 3] for further details on the theory of logic programming.

1.3.1 Basic Logical Definitions

A first order language consists of an alphabet and of a set of well-formed formulas defined on it. An alphabet contains two kinds of symbols: the logical and the non-logical ones. All the sets we consider below are countable.

The logical symbols are common to all first order languages and consist of the following sets

logical connectives:	$\neg$ (negation), $\rightarrow$ (implication), $\vee$ (disjunction), $\wedge$ (conjunction), $\leftrightarrow$ (equivalence),
variables:	an enumerable set V of variables,
quantifiers:	$\exists$ (there exists) and $\forall$ (for all),
propositional constants:	*true* and *false*,
punctuation symbols:	parentheses, comma and full stop.

The non-logical symbols determine a specific first order language and consist of the following sets

- a set Σ of *function symbols* of fixed arity,
- a set Π of *predicate symbols* of fixed arity.

A (predicate or function) symbol f with arity n will be denoted by f/n. Function symbols whose arity is 0 are called also *constant* symbols. We assume that V, Σ and Π are pairwise disjoint and do not share any symbol with the other sets mentioned above.

The set $\mathsf{T}(\Sigma, \mathsf{V})$ of *terms* of the language is inductively defined as follows.

- A variable is a term.
- A constant symbol is a term.
- If f is a function symbol with arity n and $t_1, \ldots, t_n$ are terms, then also $\mathsf{f}(t_1, \ldots, t_n)$ is a term.

$\mathsf{T}(\Sigma)$ denotes the set of ground terms (i.e., terms not containing variables). In the following, by $\mathbf{t}$ and $\mathbf{x}$ we denote tuples of, respectively, terms and *distinct* variables. Moreover, we denote by $\mathbf{t}$ both the tuple and the set of corresponding syntactic objects.

Finally we can inductively define the class of (well-formed) *formulas* of the language as follows:

- *true* and *false* are formulas.
- If p is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms, then $\mathsf{p}(t_1, \ldots, t_n)$ is a formula (called an *atomic formula* or, more simply *atom*).
- If F and G are formulas, then so are $\neg\mathsf{F}$, $\mathsf{F} \wedge \mathsf{G}$, $\mathsf{F} \vee \mathsf{G}$, $\mathsf{F} \rightarrow \mathsf{G}$ and $\mathsf{F} \leftrightarrow \mathsf{G}$.
- If F is a formula and x is a variable, then $\forall x.\, \mathsf{F}$ and $\exists x.\, \mathsf{F}$ are formulas.

An expression E is any syntactic object, i.e., terms, formulas, *etc.* A *ground* expression is a variable-free expression. In the following the set of variables that occurs in a syntactic object E is denoted $\mathit{var}(\mathsf{E})$. Moreover, by $\mathit{preds}(\mathsf{E})$ we denote the set of predicates occurring in E.

1.3.2 Unification Theory

Now we will introduce some general definitions and some results concerning unification that we need in the following. The readers interested in more details on substitutions, equations and unification can see [42, 68, 84].

A *substitution* is a mapping $\vartheta : \mathsf{V} \to \mathsf{T}(\Sigma, \mathsf{V})$, such that the set $\mathit{dom}(\vartheta) := \{x \mid \vartheta(x) \neq x\}$ (*domain of* ϑ) is finite. Substitutions which gives ground terms are called *ground*. By ε we denote the *empty substitution* (i.e., the identity function). We will use the set-theoretic notation to represent ϑ. Namely,

$$\vartheta = \{x_1/t_1, \ldots, x_n/t_n\}$$

where $\{x_1, \ldots, x_n\} := \mathit{dom}(\vartheta)$ and $t_i := \vartheta(x_i)$. A pair x_i/t_i is called a *binding*. Note that by definition of substitution, the variables $x_1, \ldots, x_n$ are different and $x_i \neq t_i$, for any $i \in [1, n]$. Let $\mathbf{x} := x_1, \ldots, x_n$ and $\mathbf{t} := t_1, \ldots, t_n$. In the following, if (for any $i \in [1, n]$) $x_i \neq t_i$, then $\{\mathbf{x}/\mathbf{t}\}$ denotes the substitution $\{x_1/t_1, \ldots, x_n/t_n\}$.

$\mathit{range}(\vartheta)$ denotes the *range* of ϑ, i.e., the set $\{y \mid x \in \mathit{dom}(\vartheta), y \in \mathit{var}(\vartheta(x))\}$. Thus, the substitution ϑ is *ground* if and only if $\mathit{range}(\vartheta) = \emptyset$. Note that $\mathit{dom}(\vartheta) \cup \mathit{range}(\vartheta) = \mathit{var}(\vartheta)$.

The *application* $\mathsf{E}\vartheta$ of ϑ to an expression E is defined as the expression obtained by simultaneously replacing each variable x in E by the corresponding term $\vartheta(x)$. $\mathsf{E}\vartheta$ is called an *instance* of E and by $\mathit{ground}(\mathsf{E})$ we denote the set of ground instances of E. If ϑ is a substitution and $W \subseteq \mathsf{V}$ is a set of variables, $\vartheta|_W$ is the restriction of ϑ to the variables W defined as follows

$$\vartheta|_W(x) := \begin{cases} x\vartheta & \text{if } x \in W \\ x & \text{if } x \notin W. \end{cases}$$

In the following, when no ambiguity arise, we will denote the substitution $\vartheta|_{\mathit{var}(\mathsf{E})}$ simply by $\vartheta|_{\mathsf{E}}$.

The *composition* $\vartheta\sigma$ of the substitutions ϑ and σ is defined as $(\vartheta\sigma)(x) := (x\vartheta)\sigma$. By using the set-theoretic notation, if $\vartheta := \{x_1/t_1, \ldots, x_n/t_n\}$ and $\sigma := \{y_1/s_1, \ldots, y_m/s_m\}$, their composition $\vartheta\sigma$ is defined by removing from the set

$$\{x_1/t_1\sigma, \ldots, x_n/t_n\sigma, y_1/s_1, \ldots, y_m/s_m\}$$

the bindings $x_i/t_i\sigma$ for which $x_i = t_i\sigma$, as well as those bindings y_j/s_j for which $y_j \in \{x_1, \ldots, x_n\}$. We recall that the composition is associative, the empty substitution ε is the neutral element and, for each expression E, $E(\vartheta\sigma) = (E\vartheta)\sigma$.

The pre-ordering $\leq$ (more general than) on substitutions is such that $\vartheta \leq \sigma$ if and only if there exists ϑ' such that $\vartheta\vartheta' = \sigma$. Similarly, given two terms t and t', we define $t \leq t'$ (t is more general than t') if and only if there exists a substitution ϑ such that $t\vartheta = t'$. The relation $\leq$ is a preorder (called *subsumption*) and by $\equiv$ we denote the associated equivalence relation (*variance*). Then two terms t and t' are *variants* if and only if t is an instance of t' and vice versa. This definition is equivalent to say that t and t' are variants if and only if there exists a renaming ρ such that $t = t'\rho$. A *renaming* is a substitution ρ for which there exists the inverse ρ^{-1}, such that $\rho\rho^{-1} = \rho^{-1}\rho = \varepsilon$.

These definitions can be extended to any kind of expression in the obvious way.

A substitution ϑ is *idempotent* if $\vartheta\vartheta = \vartheta$ or, equivalently, if $dom(\vartheta) \cap range(\vartheta) = \emptyset$. In the following, we restrict our attention to idempotent substitutions, unless explicitly stated otherwise. The set of all idempotent substitutions is denoted by *Subst*.

Unifiers and Equations

A substitution ϑ is a *unifier* of the terms t and t' if $t\vartheta = t'\vartheta$. An unifier ϑ of t and t' is called *relevant* if all the variables appearing in ϑ also appear in t or t'. An unifier of t and t' is called the *most general unifier* (*mgu*) if it is more general of any other unifier of t and t'. If $\mathbf{t} := t_1, \ldots, t_n$ and $\mathbf{s} := s_1, \ldots, s_n$, a (most general) unifier of $\mathbf{t}$ and $\mathbf{s}$ is a (most general) unifier of $(s_1, \ldots, s_n)$ and $(t_1, \ldots, t_n)$.

It is well known that the *mgu* of two terms is unique up to renaming (see for example [42]). Moreover, if two terms are unifiable, then there exists an idempotent *mgu* for them. This is formally stated by following theorem due to Robinson [87].

Theorem 1.3.1 (Unification Theorem) *[87] There exists an algorithm (called* unification algorithm*) which, for any two atoms, produces their most general unifier, if they are unifiable, and otherwise reports non-existence of an unifier.*

Since idempotent *mgus* of t_1 and t_2 are unique up to renaming, we will denote by $mgu(t_1, t_2)$ the idempotent most general unifier of t_1 and t_2 obtained by any (arbitrary but once and for all given) choice between the possibly equivalent *mgus*.

All the above definitions can be extended to other syntactic expressions in the obvious way.

An equation is an atom $s =_e t$, where s, t are terms and $=_e$ is a predicate symbol which is interpreted as the syntactic equality ($=_e$ is written in infix form to improve

the readability). If $\mathbf{s} := s_1, \ldots, s_n$ and $\mathbf{t} := t_1, \ldots, t_n$, then $\mathbf{t} =_e \mathbf{s}$ denotes the equation set $\{s_1 =_e t_1, \ldots, s_n =_e t_n\}$. If $A := p(\mathbf{s})$ and $B := p(\mathbf{t})$, $A =_e B$ is a shorthand for $\mathbf{s} =_e \mathbf{t}$. This notation is extended to conjunction of atoms in the obvious way. In the following chapters, sets and conjunctions of equations will often be identified and, when clear from the context, we will abuse notation denoting $=_e$ by $=$.

Given an equation set $E := \{\mathbf{s} =_e \mathbf{t}\}$, a (most general) unifier of E is a (most general) unifier of $\mathbf{s}$ and $\mathbf{t}$. A unifiable set of equations (terms) has an idempotent *mgu*. Well known results on idempotent *mgus* state that, if ϑ is an idempotent *mgu* of an equation set E, then ϑ is a relevant unifier of E, i.e., $var(\vartheta) \subseteq var(E)$. Any unifier ϑ for E is called *solution* if $E\vartheta$ is variable free. A set E is solvable if it has solutions. The preordering $\leq_e$ on equation sets is such that $E \leq_e E'$ if and only if the solutions of E are also solutions of E'. Two sets E, E' are called *equivalent* (denoted by $E \approx_e E'$) if they have the same solutions.

An equation set (possibly empty) is in *solved form* if it has the form $\{v_1 =_e t_1, \ldots, v_n =_e t_n\}$ and the v_i are distinct variables which do not occur in any t_j. The variables v_i are said to be *eliminable*. The set $\{v_1, \ldots, v_n\}$ is denoted by $elim(E)$. In the following, given a set of solved form equation sets $\mathcal{E}$, by $elim(\mathcal{E})$ we denote $\bigcup_{E \in \mathcal{E}} elim(E)$. If a set E is solvable then it has an equivalent solved form which is unique up to renaming. There exists an algorithm [68] which transforms any solvable equation set into an equivalent solved form equation set.

The lattice structure on idempotent substitutions [42] is isomorphic to the lattice structure on equations introduced in [68]. Therefore, we can indifferently use equations or idempotent *mgus*. The following results show the connections between the two notions we will use in the following. Given a substitution $\vartheta := \{\mathbf{x}/\mathbf{t}\}$ we define $eqn(\vartheta) := \{\mathbf{x} =_e \mathbf{t}\}$. If ϑ is an idempotent *mgu* of E, $eqn(\vartheta)$ is called the solved form of E [68]. Observe that, for any substitution ϑ, $\vartheta = mgu(eqn(\vartheta))$.

Here are some technical results on properties of substitutions and equation sets.

Theorem 1.3.2 *[68] Assume that the Herbrand universe is non-trivial (i.e., it contains at least two elements). ϑ is an idempotent mgu of E if and only if $eqn(\vartheta) \approx_e E$.*

In the following we will always implicitly consider a non-trivial Herbrand universe.

From the above result we can derive the following useful lemmata.

Lemma 1.3.3 *[13] Let E_1, E_2 be equation sets. Then there exists $\beta := mgu(E_1 \cup E_2)$ if and only if there exist $\vartheta := mgu(E_1)$ and $\delta = mgu(E_2\vartheta)$, such that $\beta = \vartheta\delta$.*

Lemma 1.3.4 *[54] Let E be an equation set and $\vartheta = mgu(E)$. Then for any substitution δ, $mgu(E\delta) = mgu(eqn(\vartheta)\delta)$.*

Corollary 1.3.5 *Let* E *be an equation set and* $\vartheta = mgu(\mathsf{E})$. *Given a renaming* ρ, *let* $\delta := \{x\rho/t\rho \mid x/t \in \vartheta\}$. *Then the following facts hold*

1. $\delta = mgu(\mathsf{E}\rho)$,
2. $\rho\delta\rho^{-1} = \vartheta$ *(and therefore* $\delta = \rho^{-1}\vartheta\rho$*).*

Proof. We prove the points separately.

Point 1 By Lemma 1.3.4 we have only to prove that $\delta = mgu(eqn(\vartheta)\rho)$. Then it is sufficient to observe that δ is an idempotent substitution and $eqn(\vartheta)\rho = eqn(\delta)$.

Point 2 Let z be a variable. By definition of δ, $\rho\delta\rho^{-1}(z) = \rho\rho^{-1}(\vartheta(z)) = \vartheta(z)$. ■

1.3.3 Logic Programs and their Classical Semantics

Throughout the thesis we will assume logic programs and goals being defined on a fixed first order language given by a non-empty *signature* Σ, a finite set Π of *predicates* and a denumerable set V of *variables*.

An *atom* (as already mentioned) is an object of the form $p(t_1, \ldots, t_n)$ where $p/n \in \Pi$ and $t_1, \ldots, t_n \in \mathsf{T}(\Sigma, \mathsf{V})$. The set of all atoms is denoted by *Atoms*. An atom is called *pure* if it is in the form $p(\mathbf{x})$.

A *program (or definite) clause* is a formula of the form $H \leftarrow B_1, \ldots, B_n$ with $n \geq 0$, where H (the *head*) and $B_1, \ldots, B_n$ (the *body*) are atoms, "$\leftarrow$" and "," denote logical implication and conjunction respectively, and all the variables are assumed to be universally quantified in front of the formula (here and in the following we omit the quantifiers). If the body is empty the clause is a *unit clause*.

A *goal* (or *negative clause*) is a formula of the form $\leftarrow A_1, \ldots, A_n$, with $n \geq 0$. When no ambiguity arises, we will denote the above goal simply by the sequence of atoms $A_1, \ldots, A_n$. If $n = 1$, the goal is called *atomic* and, by an abuse of notation, it is denoted with the corresponding atom. Moreover, if $n = 0$ the goal is called *empty* and is denoted by $\square$. The set of all goals is denoted by *Goals*. A goal is called *pure* if it contains only pure atoms which do not share variables. We denote by $\mathbf{G}$ and $\mathbf{B}$ possibly empty sequences of atoms. Hence, a clause can be simply denoted by $H \leftarrow \mathbf{B}$. By $\mathbf{B}, \mathbf{B}'$ we denote the concatenation of $\mathbf{B}$ and $\mathbf{B}'$.

A *logic program* is a finite set of (program) clauses and a *query* is the union of a goal $\mathbf{G}$ with a logic program P, here denoted by the formula $\mathbf{G}$ *in* P.

The syntax of logic programs can finally be formally defined as

$$\begin{aligned} QUERY &::= GOAL \textit{ in } PROG \\ GOAL &::= \square \mid ATOM, GOAL \end{aligned}$$

$$PROG ::= \emptyset \mid \{CLAUSE\} \cup PROG$$
$$CLAUSE ::= ATOM \leftarrow GOAL$$

where the language generated by *ATOM* is the set *Atoms*.

An *equational goal* is an object of the form $(\mathcal{E}, \mathbf{B})$ where $\mathbf{B}$ is a pure goal and $\mathcal{E}$ is a finite set of solved form equation sets such that $elim(\mathcal{E}) \subseteq var(\mathbf{B})$. $\{\emptyset\}, \mathbf{B}$ will be denoted by $\mathbf{B}$. An *equational clause* is a formula of the form $\mathsf{H} \leftarrow \mathsf{E}, \mathbf{B}$, where $(\mathsf{H}, \mathbf{B})$ is a pure goal and E is a solved form equation set such that $elim(\mathsf{E}) \subseteq var(\mathsf{H}, \mathbf{B})$. In the following, given any program clause $\mathsf{p}(\mathbf{t}) \leftarrow \mathsf{p}_1(\mathbf{t}_1), \ldots, \mathsf{p}_n(\mathbf{t}_n)$, we will consider (when needed) its equational form $\mathsf{p}(\mathbf{x}) \leftarrow \mathsf{E}, \mathsf{p}_1(\mathbf{x}_1), \ldots, \mathsf{p}_n(\mathbf{x}_n)$ where $\mathsf{E} := \{\mathbf{x} =_e \mathbf{t}, \mathbf{x}_1 =_e \mathbf{t}_1, \ldots, \mathbf{x}_n =_e \mathbf{t}_n\}$ (and $\mathbf{x}, \mathbf{x}_1, \ldots, \mathbf{x}_n$ are new distinct variables).

Traditional Declarative Semantics

The traditional declarative semantics provides the meaning of a program P without analyzing the underlying computational mechanism. Therefore, to understand the meaning of a logic program, we can use the classical model theory of first order logic, based on the notion of interpretation *à la* Tarski. Let L be the underlying first order language of P.

An *interpretation* I of a first order language L simply consists of

1. a non-empty set D (domain of the interpretation) over which the variables range,
2. an assignment of each constant in L to an element of the domain D,
3. an assignment of each function symbol of arity n in L to a mapping from D^n to D,
4. an assignment of each predicate symbol of arity n in L to a subset of D^n (i.e., to a relation on D^n).

Thus, each interpretation specifies a meaning for each symbol in the language. A *variable assignment* σ (w.r.t. I) is an assignment to each variable in L of an element in the domain of I. A variable assignment σ can be lifted homomorphically to a function, still denoted by σ and called *valuation*, which maps terms to D. It is possible to show that such a lifting is unique.

The semantics (validity) of a formula F in a variable assignment σ over I, written $\mathrm{I} \models_\sigma \mathsf{F}$, can be defined inductively on its structure as follows:

- $\mathrm{I} \models_\sigma$ *true* and not $\mathrm{I} \models_\sigma$ *false*,
- $\mathrm{I} \models_\sigma \mathsf{p}(\mathsf{t}_1, \ldots, \mathsf{t}_n)$ if and only if p_I is the relation associated with p in I and $(\sigma(\mathsf{t}_1), \ldots, \sigma(\mathsf{t}_n)) \in \mathsf{p}_\mathrm{I}$,
- $\mathrm{I} \models_\sigma \neg\mathsf{F}$ if and only if not $\mathrm{I} \models_\sigma \mathsf{F}$,

- $I \models_\sigma F \wedge G$ if and only if $I \models_\sigma F$ and $I \models_\sigma G$,
- $I \models_\sigma \forall x. F$ if and only if, for all $d \in D$, $I \models_{\sigma[d/x]} F$.[1]

The semantics of all the other connectives can be expressed by means of the semantics of $\wedge$, $\neg$ and $\forall$, by defining $F \vee G := \neg(F \wedge G)$, $F \rightarrow G := \neg F \vee G$, $F \leftrightarrow G := F \rightarrow G \wedge G \rightarrow F$ and $\exists x. F := \neg\forall x. \neg F$.

We say that F is *true* in an interpretation I ($I \models F$) if and only if it is true for each variable assignment. Moreover, we say that an interpretation I is a *model* of a set of formulas S if and only if each formula in S is true in I. A set of formulas S is *satisfiable* (or *consistent*) if and only if S has a model. Otherwise, we say that it is *unsatisfiable* (or *inconsistent*). Moreover, a query $\mathbf{G}$ *in* P is consistent (inconsistent) if the set of formulas $\{\leftarrow \mathbf{G}\} \cup P$ it is. Given two sets of formulas, S and S', we say S' is a *logical consequence* of S if and only if every model of S is also a model for S'.

When considering logic programs, the basic problem is to prove that a query $\mathbf{G}$ *in* P is unsatisfiable. According to the definition, this implies showing that every interpretation of $\mathbf{G}$ *in* P is not a model. However, this problem can be solved in a convenient way, by considering a restricted class of interpretations, namely Herbrand interpretations. *The importance of Herbrand interpretations relies in the fact that (by Theorem 1.3.6), in order to prove the unsatisfiability of a set of clauses, it is sufficient to consider only Herbrand interpretations.*

The *Herbrand Universe* $\mathcal{U}_L$ for the language L is the set of ground terms $T(\Sigma)$, while the *Herbrand Base* $\mathcal{B}_L$ of L is the set of all ground atoms of L. A *Herbrand interpretation* H is an interpretation with domain $\mathcal{U}_L$, which assigns

- each constant in L to itself,
- each function symbol f/n in L to the mapping $f_H : \mathcal{U}_L^n \rightarrow \mathcal{U}_L$, defined by $f_H(t_1, \ldots, t_n) := f(t_1, \ldots, t_n)$,
- each predicate symbol $p/n \in \Pi$ to an element p_H in $\wp(\mathcal{U}_L^n)$.

We make no restriction on the assignment of predicate symbols in L. So, for each different assignment, there is a different Herbrand interpretation. A *Herbrand model* is a Herbrand interpretation which is a model.

Each Herbrand interpretation H can be uniquely determined by a set $I \subseteq \mathcal{B}_L$, such that $p(\mathbf{t}) \in I$ if and only if $\mathbf{t} \in p_H$. Thus, from now on we can identify a Herbrand interpretation H by the set of atom determining it. Herbrand interpretations (as sets of atoms) with the subset ordering are a complete lattice.

In the following we will assume that Σ contains at least one function symbol of arity 0 (a constant) and therefore the Herbrand universe is not empty.

[1] We recall that $\sigma[d/x]$ is the valuation which differs from σ only for the assignment of x to d.

Theorem 1.3.6 (Herbrand Theorem) *A set of clauses* S *has a model if and only if it has a Herbrand model.*

As a consequence, when considering the model-theoretic semantics of logic programs, we need to look only to Herbrand models.

Herbrand models are closed under intersection. Namely, let P be a program and $\{\mathsf{M}_i\}_{i\in I}$ be a non-empty set of Herbrand models of P. Then $\bigcap_{i\in I}\mathsf{M}_i$ is a Herbrand model of P and is the *least Herbrand model*. The least Herbrand model was proposed by van Emden and Kowalski [97] as model-theoretic semantics for logic programs.

We can give an alternative fixpoint characterization of the least Herbrand model. Associated with every program there is a monotonic map, the *immediate consequence operator* T_P, which maps Herbrand interpretations to Herbrand interpretations [97, 7]. The T_P operator (associated to the logic program P) can be defined as follows. Let I be a Herbrand interpretation. Then

$$\mathsf{T}_\mathsf{P}(\mathrm{I}) := \big\{\mathsf{H} \in \mathcal{B}_\mathsf{L} \mid \mathsf{H} \leftarrow \mathbf{B} \text{ is a ground instance of a clause in } \mathsf{P} \text{ and } \mathbf{B} \subseteq \mathrm{I}\big\}.$$

The T_P mapping is continuous on the complete lattice of Herbrand interpretations. Herbrand interpretations which are models for P can be characterized in terms of T_P. Namely, given a Herbrand interpretation I, I is a model for P if and only if it is a pre-fixpoint of T_P (i.e., $\mathsf{T}_\mathsf{P}(\mathrm{I}) \subseteq \mathrm{I}$). An immediate consequence of this result (using Theorem 1.2.2) is that $\mathsf{M}_\mathsf{P} = \mathit{lfp}\,\mathsf{T}_\mathsf{P} = \mathsf{T}_\mathsf{P}{\uparrow}\omega$.

Traditional Operational Semantics

Definite clauses have a natural computational reading based on the resolution procedure. Resolution was introduced by Robinson [87] as an inference rule for clauses. The specific resolution strategy that we consider here is called *SLD-resolution* (Selection rule driven Linear resolution for Definite clauses) [7] and can be described as follows. Let $\mathbf{G} := \mathsf{A}_1, \ldots, \mathsf{A}_k$ be a goal and $\mathsf{c} := \mathsf{H} \leftarrow \mathbf{B}$ be a *input* clause. $\mathbf{G}'$ is *derived* from $\mathbf{G}$ and c (or, equivalently, $\mathbf{G}'$ is a *resolvent* of $\mathbf{G}$ and c) by using ϑ if and only if the following facts hold

1. A_m (with $1 \leq m \leq k$) is the *selected atom* in $\mathbf{G}$,
2. $\vartheta = \mathit{mgu}(\mathsf{A}_m, \mathsf{H})$,
3. $\mathbf{G}' := (\mathsf{A}_1, \ldots, \mathsf{A}_{m-1}, \mathbf{B}, \mathsf{A}_{m+1}, \ldots, \mathsf{A}_k)\vartheta$.

Given a goal $\mathbf{G}$ and a program P, an *SLD-derivation* (or simply a derivation) $\mathtt{d}$ of $\mathbf{G}$ *in* P consists of

1. a (possibly infinite) sequence of goals $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2, \ldots$ called resolvents, together with

2. a sequence $c_1, c_2, \ldots$ of variants of clauses in P which are *renamed apart*[2] and
3. a sequence $\vartheta_1, \vartheta_2, \ldots$ of idempotent *mgus*[3],

such that $\mathbf{G}_0 = \mathbf{G}$ and, for $i \geq 1$, each $\mathbf{G}_i$ is derived from $\mathbf{G}_{i-1}$ and c_i by using ϑ_i.

An *SLD-refutation* of $\mathbf{G}$ *in* P is a finite *SLD*-derivation of $\mathbf{G}$ *in* P which has the empty goal $\square$ as the last goal in the derivation. If $\vartheta_1, \ldots, \vartheta_n$ are the *mgus* used in the refutation of $\mathbf{G}$ *in* P, we say that $\vartheta := (\vartheta_1 \cdots \vartheta_n)|_{\mathbf{G}}$ is a *computed answer substitution* (or simply a computed answer) of $\mathbf{G}$ *in* P. Moreover, given a derivation d of $\mathbf{G}$ *in* P, with $\mathbf{B}$ as last goal and $\vartheta_1, \ldots, \vartheta_n$ as the used *mgus*, we say that $\vartheta := (\vartheta_1 \cdots \vartheta_n)|_{\mathbf{G},\mathbf{B}}$ is a *(partial) computed answer substitution* of d. In case d is a refutation, this definition boils down to the previous definition of computed answer for $\mathbf{G}$ *in* P.

Given a derivation d, *first*(d) and *last*(d) (if d is finite) are respectively the first and the last goal of d. By *clauses*(d) we denote the sequence of clauses of d and by *answer*(d) we denote the (partial) computed answer substitution of d. Moreover, *length*(d) denotes the length of the derivation d, i.e., the number of the *SLD*-derivation steps. By an abuse of notation, we denote a zero-length derivation of $\mathbf{G}$ by $\mathbf{G}$ itself. Hence, when the length of the derivation is 0, *first*$(d) =$ *last*$(d) = \mathbf{G}$ and *answer*$(d) = \varepsilon$. Finally, *prefix*(d) is the set of all *SLD*-derivations which are prefixes of d.

Example 1.3.7

Consider the derivations

$$d_0 := p(x, y)$$

$$d_1 := p(x, y) \xrightarrow[p(v,w) \leftarrow q(v,w)]{\{v/x, w/y\}} q(x, y)$$

$$d_2 := q(x, y) \xrightarrow[q(t,u) \leftarrow n(f(s),t)]{\{t/x, u/y\}} n(f(s), x) \xrightarrow[n(r,a) \leftarrow]{\{r/f(s), x/a\}} \square$$

$$d_3 := p(x, y) \xrightarrow[p(g(a),b) \leftarrow]{\{x/g(a), y/b\}} \square$$

$$d_4 := q(f(x), z) \xrightarrow[q(t,a) \leftarrow m(g(s),t)]{\{t/f(x), z/a\}} m(g(s), f(x)) \xrightarrow[m(r,f(r)) \leftarrow]{\{r/g(s), x/g(s)\}} \square$$

[2] i.e., such that c_i does not share any variable with $\mathbf{G}_0, c_1, \ldots, c_{i-1}$.

[3] The *mgus* used in *SLD*-derivations are not always assumed to be idempotent. Note that, since idempotent *mgus* are also relevant, the previous "static" definition of renaming apart of clauses is sufficient (see [3]). Such a static renaming would not be correct when non relevant *mgus* are used. For example, if the (non relevant) *mgu* of the goal $\mathbf{G}$ and of the head of clause c_1 introduces a new variable $x \notin var(\mathbf{G}) \cup var(c_1)$, according to the previous definition, c_2 could (incorrectly) contain the variable x. In this case, a correct notion of renaming apart should involve also the previously *mgus* used in the derivation. Therefore, such a notion could not be stated independently of the derivation.

$$\mathsf{d}_5 := \mathsf{n}(\mathsf{x}) \xrightarrow[\mathsf{n}(\mathsf{t}) \leftarrow \mathsf{p}(\mathsf{t},\mathsf{t})]{\{\mathsf{t}/\mathsf{x}\}} \mathsf{p}(\mathsf{x},\mathsf{x})$$

Then the following holds.

$$\begin{aligned}
\mathit{first}(\mathsf{d}_1) &= \mathsf{p}(\mathsf{x},\mathsf{y}) & \mathit{last}(\mathsf{d}_1) &= \mathsf{q}(\mathsf{x},\mathsf{z}) \\
\mathit{first}(\mathsf{d}_4) &= \mathsf{q}(\mathsf{f}(\mathsf{x}),\mathsf{z}) & \mathit{last}(\mathsf{d}_4) &= \square \\
\mathit{answer}(\mathsf{d}_0) &= \varepsilon & \mathit{answer}(\mathsf{d}_1) &= \varepsilon \\
\mathit{answer}(\mathsf{d}_2) &= \{\mathsf{x}/\mathsf{a}\} & \mathit{answer}(\mathsf{d}_3) &= \{\mathsf{x}/\mathsf{g}(\mathsf{a}),\mathsf{y}/\mathsf{b}\} \\
\mathit{answer}(\mathsf{d}_4) &= \{\mathsf{x}/\mathsf{g}(\mathsf{s}),\mathsf{z}/\mathsf{a}\} & \mathit{answer}(\mathsf{d}_5) &= \varepsilon \\
\mathit{length}(\mathsf{d}_0) &= 0 & \mathit{length}(\mathsf{d}_2) &= 2
\end{aligned}$$

Moreover,

$$\begin{aligned}
\mathit{clauses}(\mathsf{d}_1) &= [\mathsf{p}(v,w) \leftarrow \mathsf{q}(v,w)] \\
\mathit{clauses}(\mathsf{d}_2) &= [\mathsf{q}(\mathsf{t},\mathsf{u}) \leftarrow \mathsf{n}(\mathsf{f}(\mathsf{s}),\mathsf{t});\ \ \mathsf{n}(\mathsf{r},\mathsf{a}) \leftarrow] \\
\mathit{clauses}(\mathsf{d}_4) &= [\mathsf{q}(\mathsf{t},\mathsf{a}) \leftarrow \mathsf{m}(\mathsf{g}(\mathsf{s}),\mathsf{t});\ \ \mathsf{m}(\mathsf{r},\mathsf{f}(\mathsf{r})) \leftarrow]
\end{aligned}$$

and

$$\begin{aligned}
\mathit{prefix}(\mathsf{d}_1) &= \big\{\mathsf{p}(\mathsf{x},\mathsf{y});\ \ \mathsf{d}_1\big\} \\
\mathit{prefix}(\mathsf{d}_2) &= \big\{\mathsf{q}(\mathsf{x},\mathsf{y});\ \ \mathsf{q}(\mathsf{x},\mathsf{y}) \xrightarrow[\mathsf{q}(\mathsf{t},\mathsf{u}) \leftarrow \mathsf{n}(\mathsf{f}(\mathsf{s}),\mathsf{t})]{\{\mathsf{t}/\mathsf{x},\mathsf{u}/\mathsf{y}\}} \mathsf{n}(\mathsf{f}(\mathsf{s}),\mathsf{x});\ \ \mathsf{d}_2\big\} \\
\mathit{prefix}(\mathsf{d}_4) &= \big\{\mathsf{q}(\mathsf{f}(\mathsf{x}),\mathsf{z});\ \ \mathsf{q}(\mathsf{f}(\mathsf{x}),\mathsf{z}) \xrightarrow[\mathsf{q}(\mathsf{t},\mathsf{a}) \leftarrow \mathsf{m}(\mathsf{g}(\mathsf{s}),\mathsf{t})]{\{\mathsf{t}/\mathsf{f}(\mathsf{x}),\mathsf{z}/\mathsf{a}\}} \mathsf{m}(\mathsf{g}(\mathsf{s}),\mathsf{f}(\mathsf{x}));\ \ \mathsf{d}_4\big\}
\end{aligned}$$

Note that $\mathit{card}(\mathit{prefix}(\mathsf{d})) = \mathit{length}(\mathsf{d}) + 1$.

According to the definition of *SLD*-derivation the following two choices are performed at each step in the resolvent construction:

1. choice of the selected atom in the goal,
2. choice of the input clause in the program.

The choice of the input clause can cause some problems, related to the possibly lack of completeness of the resulting *SLD*-refutations, as shown in [75]. In the following we will assume that any program clause can be applied to perform *SLD*-derivations.

The choice of the selected atom in the goal is done according to the *selection rule* R, which is usually defined (see [3]) as a function which, when applied to a "history" containing the goal and all the clauses and the *mgus* used in the derivation $\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_i$, returns an atom in $\mathbf{G}_i$. Such an atom is the *selected atom* in $\mathbf{G}_i$.

Given a selection rule R, an *SLD*-derivation via R is an *SLD*-derivation where all the selections of atoms in the resolvents are computed according to R. An R-computed answer of $\mathbf{G}$ *in* P is a computed answer of an *SLD*-refutation via R. There is an important theorem about independence from selection rule which states that if the substitution ϑ is a computed answer of $\mathbf{G}$ *in* P, then (for any selection rule R) there exists an R-computed answer σ of $\mathbf{G}$ *in* P such that $\mathbf{G}\vartheta$ and $\mathbf{G}\sigma$ are variants. Thus, the choice of the selection rule only affects *termination* (and in general how efficiently a computed answer is found). A rule R is *fair* if any *SLD*-derivation via R is fair, i.e., if it is either finite or every atom appearing in it is eventually selected.

When searching for a refutation of a goal, *SLD*-derivations are constructed with the aim of generating the empty goal. The totality of these derivations forms a *search space*. One way of organizing this search space is by grouping *SLD*-derivations according to the selection rule used. This brings up to the concept of *SLD*-tree.

An *SLD-tree* of $\mathbf{G}$ *in* P via R is the prefix tree of all *SLD*-derivations of $\mathbf{G}$ *in* P via R. The *prefix tree* of a set of possibly infinite sequences W (all starting with the same element) is a tree which has as nodes all the elements of the sequences in W (without repetitions) and whose branches link nodes which are consecutive elements of a sequence in W. In everyday practice, input clauses and resolvent substitutions are moved from the nodes to the label of the incoming arc of the node. Thus, the root node in an *SLD*-tree for $\mathbf{G}$ *in* P via R is $\mathbf{G}$ and every node in this tree is a goal whose descendant are all its resolvents with (variants of) clauses of P, where the selected atom is chosen according to R.

We call an *SLD*-tree *successful* if it contains the empty goal and *finitely failed* if it is finite and not successful.

In the following, for the sake of simplicity, we will consider the PROLOG selection rule, which always selects the leftmost atom and is then called *leftmost*. All our results can be generalized to skeleton rules [54] (the leftmost rule is a particular case of skeleton rules). *SLD*-derivations via the leftmost rule are called *LD-derivations*. In the following $\mathbf{G} \xrightarrow[\mathsf{c}]{\vartheta} \mathbf{B}$ denotes an *LD*-derivation step of $\mathbf{G}$ which uses the input clause c, where ϑ is the computed *mgu* and $\mathbf{B}$ is the resolvent. Moreover, by

$$\mathsf{d} := \mathbf{G} \xrightarrow[\mathsf{c}_1]{\vartheta_1} \cdots \xrightarrow[\mathsf{c}_n]{\vartheta_n} \mathbf{G}_n$$

we denote a (finite) *LD*-derivation of $\mathbf{G}$ *in* P, where any c_i is a renamed clause of P. We also denote by $\mathbf{G} \xrightarrow[\mathsf{P}]{\vartheta}{}^{*} \mathbf{B}$ a finite *LD*-derivation of $\mathbf{G}$ *in* P, where ϑ is the (partial) computed answer substitution and $\mathbf{B}$ is the last resolvent. In the following, to simplify the notation, $\mathbf{G}_0 \xrightarrow[\mathsf{c}_1,\ldots,\mathsf{c}_n]{\vartheta} \mathbf{G}_n$ will represent the derivation $\mathbf{G}_0 \xrightarrow[\mathsf{c}_1]{\vartheta_1} \cdots \xrightarrow[\mathsf{c}_n]{\vartheta_n} \mathbf{G}_n$ such that $\vartheta = \vartheta_1 \cdots \vartheta_n$.

SLD-derivations enjoy an interesting (technical) property, stated by the following lemma, which will be useful in the sequel. The proof of it is essentially the same of that given for Lemma 2.2 in [54], except for a slight difference in the notation.

Lemma 1.3.8 *[54] Let* $\mathbf{G}$ *be a goal and* δ *be an idempotent substitution.*

1. *For any derivation* $\mathbf{G}\delta \xrightarrow[c_1,\ldots,c_m]{\vartheta} \mathbf{G}_m$ *such that* $var(c_1,\ldots,c_m) \cap var(\mathbf{G}) = \emptyset$, *there exist a derivation* $\mathbf{G} \xrightarrow[c_1,\ldots,c_m]{\vartheta'} \mathbf{G}'_m$ *and a substitution* $\delta' = mgu(\mathbf{G}\delta, \mathbf{G}\vartheta')$, *such that* $\delta|_{\mathbf{G}}\vartheta = \vartheta'\delta'$ *and* $\mathbf{G}_m = \mathbf{G}'_m\delta'$.

2. *Conversely, for any derivation* $\mathbf{G} \xrightarrow[c_1,\ldots,c_m]{\vartheta'} \mathbf{G}'_m$ *such that* $var(c_1,\ldots,c_m) \cap var(\mathbf{G}\delta) = \emptyset$ *and* $\mathbf{G}\delta$ *and* $\mathbf{G}\vartheta'$ *are unifiable, then there exists a derivation* $\mathbf{G}\delta \xrightarrow[c_1,\ldots,c_m]{\vartheta} \mathbf{G}_m$ *and a substitution* $\delta' = mgu(\mathbf{G}\delta, \mathbf{G}\vartheta')$, *such that* $\delta|_{\mathbf{G}}\vartheta = \vartheta'\delta'$ *and* $\mathbf{G}_m = \mathbf{G}'_m\delta'$.

Up to now we have shown how *SLD*-derivation gives a proof-theory for logic programs. The *operational semantics* of logic programs can then be defined naturally in terms of such an inference rule. Namely, given a program P on the language L, we can define the *success set* of a program P as the set of all (ground) atoms A in the Herbrand Base $\mathcal{B}_L$, such that A *in* P has a refutation, i.e.,

$$\mathcal{O}_{ss}[\![P]\!] = \{A \in \mathcal{B}_L \mid A \xrightarrow[P]{\varepsilon}{}^{*} \Box\}.$$

Classical Results

The following result shows the equivalence between the traditional operational definition (success set) and the traditional declarative one (least Herbrand model).

Theorem 1.3.9 (Success set theorem) *[7] The success set of a program* P *is equal to its least Herbrand model* ($\mathcal{O}_{ss}[\![P]\!] = M_P$).

The main classical results on *SLD*-resolution concern its soundness and completeness w.r.t. semantics.

Theorem 1.3.10 (Soundness) *[16] Let* P *be a program,* $\mathbf{G} := A_1,\ldots,A_k$ *be a goal and* $\vartheta_1,\ldots,\vartheta_n$ *be the sequence of mgus computed in an SLD-refutation of* $\mathbf{G}$ *in* P. *Then* $(A_1 \wedge \cdots \wedge A_k)\vartheta_1\cdots\vartheta_n$ *is a logical consequence of* P.

As a corollary of the previous theorem, if there exists an *SLD*-refutation of $\mathbf{G}$ *in* P then $\mathbf{G}$ *in* P is inconsistent. The converse of this corollary gives a first completeness result. It is due to Hill [63] even if the proof is due to Apt and van Emden [7].

Theorem 1.3.11 (Completeness) *[7] Let* P *be a program and* $\mathbf{G}$ *be a goal. If* $\mathbf{G}$ *in* P *is inconsistent then there exists an SLD-refutation of* $\mathbf{G}$ *in* P.

A major completeness result can be obtained by considering computed answer substitutions. Let $\mathbf{G} := A_1, \ldots, A_k$ be a goal and ϑ be a substitution. We say that ϑ is a *correct answer* for $\mathbf{G}$ *in* P if and only if $dom(\vartheta) \subseteq var(\mathbf{G})$ and $\mathsf{P} \models (A_1 \wedge \cdots \wedge A_k)\vartheta$.

Theorem 1.3.12 *[16] Let* P *be a program and* $\mathbf{G} := A_1, \ldots, A_k$ *be a goal. If* ϑ *is a correct answer for* $\mathbf{G}$ *in* P*, then there exists a computed answer* σ *of* $\mathbf{G}$ *in* P *such that* $\mathbf{G}\sigma$ *is more general than* $\mathbf{G}\vartheta$.

We can generalize Theorems 1.3.11 and 1.3.12 by showing that their results do not depend on the choice of the selection rule.

Theorem 1.3.13 (Strong Completeness) *[63] Let* P *be a program,* $\mathbf{G}$ *be a goal and* R *be a selection rule. If* $\mathbf{G}$ *in* P *is inconsistent then there exists an SLD-refutation of* $\mathbf{G}$ *in* P *via* R.

Theorem 1.3.14 *[16] Let* P *be a program,* $\mathbf{G} := A_1, \ldots, A_k$ *be a goal and* R *a selection rule. If* ϑ *is a correct answer for* $\mathbf{G}$ *in* P*, then there exists a* R*-computed answer* σ *of* $\mathbf{G}$ *in* P *such that* $\mathbf{G}\sigma$ *is more general than* $\mathbf{G}\vartheta$.

Equational Version of *SLD*-derivations

In order to prove several of the technical results in the thesis we introduce now an equational version [98] of *SLD*-derivations, denoted by $\xrightarrow[c]{eq}$, which uses equations instead of idempotent *mgus*. The equivalence between these two different versions of *SLD*-derivation can be easily proved by using the previously mentioned isomorphism between substitutions and equations.

An *equational goal* is a conjunction $\mathsf{E}, \mathbf{G}$ where E is a conjunction of equations and $\mathbf{G}$ is a conjunction of non-equational atoms. We also assume that the equational part of queries is never taken into account by the selection rule (i.e., the selection rule as a function does not have the equational parts of resolvents neither in its domain nor in its range). Hence, for example, the leftmost rule selects the atom $p(a, x)$ in the goal $s = t, p(a, x), q(x, y)$.

Equational derivations (via the leftmost selection rule) are formally defined as follows. Let us consider the equational goal $\mathsf{E}, A_1, \ldots, A_n$ and let $c := H \leftarrow \mathbf{B}$ be a renamed apart clause such that $\mathsf{E}' := \mathsf{E} \cup \{A_1 =_e H\}$ is unifiable. Then we have an *equational derivation step* $\mathsf{E}, A_1, \ldots, A_n \xrightarrow[c]{eq} \mathsf{E}', \mathbf{B}, A_2, \ldots, A_n$. Equational derivations are obtained from derivation steps in the usual way. An *equational*

refutation is a finite equational derivation which has only equational atoms in the last resolvent.

As pointed out in [98], any *SLD*-derivation can be transformed in an equivalent equational one, which uses the same clauses. If ϑ is the idempotent *mgu* of E, then the (idempotent) *mgu* ϑ' of $\mathsf{E}'\vartheta$ is the computed substitution corresponding to the same step of the original *SLD*-derivation. The equivalence is formally stated by the following lemma. To simplify the notation, $\mathsf{E}_0, \mathbf{G}_0 \xrightarrow[c_1,\ldots,c_n]{eq} \mathsf{E}_n, \mathbf{G}_n$ will represent the equational derivation $\mathsf{E}_0, \mathbf{G}_0 \xrightarrow[c_1]{eq} \cdots \xrightarrow[c_n]{eq} \mathsf{E}_n, \mathbf{G}_n$.

Lemma 1.3.15 *[54] Let* $\mathbf{G}$ *be a goal. There exists a derivation* $\mathbf{G} \xrightarrow[c_1,\ldots,c_n]{\vartheta} \mathbf{A}$ *if and only if there exists an equational derivation* $\mathbf{G} \xrightarrow[c_1,\ldots,c_n]{eq} \mathsf{E}, \mathbf{B}$ *such that* $\vartheta = mgu(\mathsf{E})$ *and* $\mathbf{A} = \mathbf{B}\vartheta$.

1.3.4 The *s*-semantics Approach

The aim of the s-semantics approach is modeling the observable behaviors of logic programs (for a complete survey see [13]). It is worth noting that the aim of the approach is not defining a new (artificial and futile) notion of model. The matter is that the traditional declarative semantics is unsatisfactory because it characterizes the logical properties only. So it is not very useful from the programming point of view. For instance, the standard bottom-up semantics, being defined on ground atoms only, does not allow us to model correctly the concept of logical variable which is essential in describing properties like types and ground dependencies. A satisfactory solution to the simple case of positive logic programs is needed to gain a better understanding of more practical languages, such as real PROLOG and its purely declarative counterparts.

The fundamental observable property which is taken into account is the *computed answer substitutions* which induces the program equivalence $\mathsf{P}_1 \approx_{ca} \mathsf{P}_2$ defined as $\mathbf{G}$ *in* P_1 has the same (modulo variance) computed answer substitutions as $\mathbf{G}$ *in* P_2, for any goal $\mathbf{G}$.

The semantics modeling computed answers (the s-*semantics* [45, 46, 13]) is defined on a generalization of Herbrand interpretation. This semantics is both simpler than the traditional denotational one and more typical for logic programming. Moreover, since it is able to deal with logical variables, it allows us to extend the scope of the bottom-up approach to analysis problems[4].

[4]Indeed, we will obtain two semantics modeling "groundness dependencies of computed answer substitutions" and "computed *depth*(k)-answer substitutions" (Sections 3.5.1 and 3.5.2) by abstracting our (equivalent) reconstruction of the s-semantics (Section 3.4.1).

The denotations are defined by *syntactic* objects, as in the case of Herbrand interpretations. Denotations (called π-*interpretations*) are not interpretations in the conventional mathematical logic sense. Namely, a π-interpretation I is a set of (non necessarily ground) atoms, i.e., $I \subseteq \mathit{Atoms}$. As in the case of the van Emden and Kowalski semantics, denotations can be computed both by a top-down construction (a success set) and by a bottom-up construction (the least fixpoint of suitable continuous immediate consequence operators on π-interpretations).

The top-down denotation can be obtained by taking the behavior (the computed answers) of pure atomic goals modulo variance. Namely,

$$\mathcal{O}_{\mathrm{ca}}[\![P]\!] := \big\{p(\mathbf{x})\vartheta \in \mathit{Atoms} \mid p(\mathbf{x}) \xrightarrow[P]{\vartheta}{}^{*} \Box\big\}/_{\equiv}$$

and actually models computed answer substitutions since $P_1 \approx_{\mathrm{ca}} P_2$ if and only if $\mathcal{O}_{\mathrm{ca}}[\![P_1]\!] = \mathcal{O}_{\mathrm{ca}}[\![P_2]\!]$. Furthermore, the behavior of any (possibly conjunctive) goal $\mathbf{G}$ can be derived from $\mathcal{O}_{\mathrm{ca}}[\![P]\!]$ by "executing" $\mathbf{G}$ in $\mathcal{O}_{\mathrm{ca}}[\![P]\!]$. Namely,

Theorem 1.3.16 *[44, 45] Let* $\mathbf{G} := A_1, \ldots, A_n$ *be a goal. Then* $\mathbf{G} \xrightarrow[P]{\vartheta}{}^{*} \Box$ *if and only if there exist (renamed apart) atoms* $A'_1, \ldots, A'_n \in \mathcal{O}_{\mathrm{ca}}[\![P]\!]$ *such that* $\vartheta = \delta\rho$, *where* $\delta := \mathit{mgu}((A_1, \ldots, A_n), (A'_1, \ldots, A'_n)|_{\mathbf{G}}$ *and* ρ *is a renaming.*

This property is a kind of *AND-compositionality*, i.e., it shows that $\mathcal{O}_{\mathrm{ca}}[\![P]\!]$ provides a denotation which can actually be used to simulate the program execution for any goal. This is also the key property which allows us to use abstractions of the semantics for goal-independent abstract interpretation.

In [13] it is shown that the classical success set semantics is not correct with respect to computed answers and that the correctness cannot be achieved by just using interpretations consisting of sets of non-ground atoms. In fact, also the *atomic logical consequence semantics* [16, 47] does not correctly model the computed answers. The atomic logical consequence semantics (which is also called c-semantics in [45]) is defined by the top-down denotation

$$\mathcal{O}_{\mathrm{lc}}[\![P]\!] := \big\{A \in \mathit{Atoms} \mid A \xrightarrow[P]{\varepsilon}{}^{*} \Box\big\}.$$

The s-semantics fixpoint denotation is $\mathcal{F}_{\mathrm{ca}}[\![P]\!] := \mathit{lfp}\, T_P^{\mathrm{ca}}$, where

$$T_P^{\mathrm{ca}}(I) = \big\{H\vartheta \in \mathit{Atoms} \mid H \leftarrow \mathbf{B} \in P,\ \mathbf{B}' \subseteq I,\ \vartheta = \mathit{mgu}(\mathbf{B}, \mathbf{B}')\big\}$$

Note that T_P^{ca} is different from the standard T_P operator [97] in that it derives instances of the clause heads by *unifying* the clause bodies with atoms in the current π-interpretation, rather than by taking all the possible ground instances. In other words, T_P^{ca} defines a bottom-up inference based on the same rule (unification) which is used by top-down *SLD*-resolution.

Another related useful property of the s-semantics is its independence from the language. This means that the denotation defined by $\mathcal{O}_{\mathrm{ca}}[\![\mathsf{P}]\!]$ is not affected by the choice of the language signature. In principle, the language signature affects the domain of π-interpretations *Atoms* and, since $\mathcal{O}_{\mathrm{ca}}[\![\mathsf{P}]\!]$ is a subset of *Atoms*, it might also be affected. This is indeed not the case [70], since if L_{P} is the language underlying program P, then $\mathcal{O}_{\mathrm{ca}}[\![\mathsf{P}]\!]^{(\mathsf{L}_{\mathsf{P}})} = \mathcal{O}_{\mathrm{ca}}[\![\mathsf{P}]\!]^{(\mathsf{L}'_{\mathsf{P}})}$ for any extension L'_{P} of L_{P}. $\mathcal{O}_{\mathrm{ca}}[\![\mathsf{P}]\!]^{(\mathsf{L})}$ denotes the s-semantics of P for a given language L.

This shows the *language independence property*. This property also means that the denotation of a program *fragment* is determined by the symbols it contains without considering the entire underlying alphabet. The advantage of enjoying such a property becomes more tangible in the case where we wish to actually *compute* program denotations – as in the case of semantic based program analysis and diagnosis. Note that the same property does not hold for other variable-based semantics, such as those in [16, 48].

1.4 Abstract Interpretation

In this section we give the basic notations and concepts of approximation theory in semantics as firstly developed in [32]. For the terminology not explicitly shown and for a more motivated introduction the reader can consult [32, 34, 1, 82].

1.4.1 Closures on Complete Lattices

Closures play a fundamental role in semantics and approximation theory [34]. In the following we recall some basic notions on closure theory that will be useful throughout the thesis. For a more complete treatment of the subject see [10, 33]. A *closure operator* on a complete lattice $(\mathsf{L}, \leq)$ is an operator $\rho : \mathsf{L} \to \mathsf{L}$ such that, for each $x, y \in \mathsf{L}$,

$$x \leq \rho(x) \qquad \text{(extensivity)}$$
$$x \leq y \Longrightarrow \rho(x) \leq \rho(y) \qquad \text{(monotonicity)}$$
$$\rho(\rho(x)) = \rho(x) \qquad \text{(idempotence)}$$

Let $(\mathsf{L}, \leq)$ be a complete lattice. In the following we enumerate some basic properties of closure operators on L. Let ρ be an upper closure operator on $(\mathsf{L}, \leq)$.

- For all $x \in \mathsf{L}$, the set $\{y \in \rho(\mathsf{L}) \mid x \leq y\}$ is not empty and $\rho(x)$ is the least element.
- The image $\mathsf{R} := \rho(\mathsf{L})$ of L by ρ is a complete lattice $(\mathsf{R}, \leq)$, such that $\bigsqcup_{\mathsf{R}}(X) = \rho(\bigsqcup_{\mathsf{L}}(X))$ and $\bigsqcap_{\mathsf{R}}(X) = \bigsqcap_{\mathsf{L}}(X)$.

- ρ is a *quasi-complete-join-morphism*. Namely, for each $X \subseteq L$, $\rho(\bigsqcup(X)) = \rho(\bigsqcup(\rho(X)))$.
- Let $R \subseteq L$ and $\rho : L \to R$ such that, for any $x \in L$, $\rho(x)$ is the least element in $\{y \in R \mid x \leq y\}$. Then ρ is an upper closure operator on $(L, \leq)$ and $R := \rho(L)$.
- Let $uco(L)$ be the set of all upper closure operators on L. Then $(uco(L), \preceq)$ is a complete lattice, where $\preceq$ is defined as follows. For each $\rho, \rho' \in uco(L)$,

$$\rho \preceq \rho' \iff \forall x \in L.\, \rho(x) \leq \rho'(x).$$

1.4.2 Galois Insertions and Abstract Interpretation

Abstract Interpretation is a theory developed to reason about the abstraction relation between two different semantics (the *concrete* and the *abstract* semantics). The idea of approximating program properties by evaluating a program on a simpler domain of descriptions of "concrete" program states goes back to the early 70's. The inspiration was that of approximating properties from the exact (concrete) semantics into an approximate (abstract) semantics, that explicitly exhibits a structure (e.g., ordering) which is somehow present in the richer concrete structure associated to program execution.

The guiding idea is to relate the concrete and the abstract interpretation of the calculus by a pair of functions, *abstraction* α and *concretization* γ, which form a Galois connection. Galois connections are used to formalize this relation between abstract and concrete meaning of a computation. This notion has been introduced in [83] to discuss a general type of correspondence between structures occurring in a great variety of mathematical theories. *Galois connections can be defined on preordered sets. However in this thesis we restrict our attention to complete lattices.*

Let $(C, \sqsubseteq)$ (concrete domain) be the domain of the concrete semantics, while $(A, \leq)$ (abstract domain) be the domain of the abstract semantics. The partial order relations reflect an approximation relation. Since in approximation theory a partial order specifies the precision degree of any element in a poset, it is obvious to assume that if α is a mapping associating an abstract object in $(A, \leq)$ for any concrete element in $(C, \sqsubseteq)$ then the following holds: if $\alpha(x) \leq y$, then y is also a correct, although less precise, abstract approximation of x. The same argument holds if $x \sqsubseteq \gamma(y)$. Then y is also a correct approximation of x, although x provides more accurate information than $\gamma(y)$. This gives rise to the following formal definition.

Definition 1.4.1 (Galois Insertion) *Let* $(C, \sqsubseteq)$ *and* $(A, \leq)$ *be two posets (the concrete and the abstract domain). A* Galois Connection $\langle \alpha, \gamma \rangle : (C, \sqsubseteq) \rightleftharpoons (A, \leq)$ *is a pair of maps* $\alpha : C \to A$ *and* $\gamma : A \to C$ *such that*

1. *α and γ are monotonic,*
2. *for each $x \in C$, $x \sqsubseteq (\gamma \circ \alpha)(x)$ and*
3. *for each $y \in A$, $(\alpha \circ \gamma)(y) \leq y$.*

Moreover, a Galois Insertion *(of A in C) $\langle \alpha, \gamma \rangle : (C, \sqsubseteq) \rightleftharpoons (A, \leq)$ is a Galois connection where $\alpha \circ \gamma = Id_A$.*

Property 2 is called *extensivity* of $\gamma\alpha$. The map α (γ) is called the *lower* (*upper*) *adjoint* or *abstraction* (*concretization*) in the context of abstract interpretation.

The following basic properties are satisfied by any Galois insertion.

1. γ is injective if and only if α is surjective if and only if $\alpha \circ \gamma = Id_A$.
2. $\gamma \circ \alpha$ is an upper closure operator in $(C, \sqsubseteq)$.
3. α is additive and γ is co-additive.
4. Upper and lower adjoints uniquely determine each other. Namely,
$$\gamma = \lambda y. \bigsqcup_C \{x \in C \mid \alpha(x) \sqsubseteq y\}, \qquad \alpha = \lambda x. \bigsqcap_A \{y \in A \mid x \leq \gamma(y)\}.$$
5. α is an isomorphism from $(\gamma\alpha)(C)$ to A, having γ as its inverse.

The last properties characterize the ability of Galois insertions to formalize the notion of "machine-representable" abstractions. An abstract domain is isomorphic (up to representation) to an upper closure operator of the concrete domain of the computation. Thus, in principle, we can handle abstract computations as concrete computations on the complete lattice which is the image of the upper closure operator $\gamma \circ \alpha$. However, machine representable abstractions often result to be more intuitive and provides better experimental results in efficient implementations.

A straightforward consequence of the latter observation is that abstract interpretations can be formalized in a hierarchical framework. Abstract domains can be partially ordered using the ordering on the corresponding closure operators on the concrete domain of computation. The lattice of abstract interpretations is then the lattice of closure operators over the concrete domain. As observed in [83] the composition of upper closure operators is not (in general) an upper closure operator. However, an abstract domain can be designed by successive approximations. Let $(L, \leq)$ be a complete lattice, ρ be an upper closure operators on $(L, \leq)$ and η be an upper operator on $\rho(L)$. Then $\eta \circ \rho$ is an upper closure operator on $(L, \leq)$. In view of the compositional design of abstract interpretations we have that the composition of Galois insertions is a Galois insertion. Several techniques can be used

to systematically derive new abstract interpretations from a given set of abstract domains [34, 35]. We do not address these techniques because they are outside the scope of this thesis.

When, in a Galois connection $\langle \alpha, \gamma \rangle$, γ is not injective, several distinct elements of the abstract domain $(A, \leq)$ have the same meaning (by γ). This is usually considered useless [34]; thus a Galois insertion can always be forced by considering a more concise abstract domain $(A/_{\overset{\gamma}{\approx}}, \leq_{\overset{\gamma}{\approx}})$, such that for each $x, y \in A : x \overset{\gamma}{\approx} y \iff \gamma(x) = \gamma(y)$.

1.4.3 Correctness and Precision of Abstract Semantic Functions

Given a concrete semantics and a Galois insertion between the concrete and the abstract domain, we want to define an abstract semantics. The theory requires the concrete semantics to be the least fixpoint of a semantic function $F : C \to C$. The abstract semantic function $\tilde{F} : A \to A$ is *correct* if $\forall x \in C.\, F(x) \sqsubseteq \gamma(\tilde{F}(\alpha(x)))$.

It is often the case (at least in this thesis) that F is in turn defined as composition of "primitive" operators. Let $f : C^n \to C$ be one such an operator and assume that $\tilde{f}$ is its abstract counterpart. Then $\tilde{f}$ is *(locally) correct* w.r.t. f if $\forall x_1, \ldots, x_n \in C.\, f(x_1, \ldots, x_n) \sqsubseteq \gamma(\tilde{f}(\alpha(x_1), \ldots, \alpha(x_n)))$. The local correctness of all the primitive operators implies the global correctness. Hence, we can define an abstract semantics by defining locally correct abstract primitive semantic functions. An abstract computation is then related to the concrete computation, simply by replacing the concrete operators by the corresponding abstract operators. According to the theory, for each operator f, there exists an optimal (most precise) locally correct abstract operator $\tilde{f}$ defined as $\tilde{f}(y_1, \ldots, y_n) = \alpha(f(\gamma(y_1), \ldots, \gamma(y_n)))$. However, the composition of optimal operators is not necessarily optimal.

The abstract operator $\tilde{f}$ is *precise*[5] if

$$\forall x_1, \ldots, x_n \in C.\, \alpha(f(x_1, \ldots, x_n)) = \tilde{f}(\alpha(x_1), \ldots, \alpha(x_n))$$

which is equivalent to $\alpha(f(x_1, \ldots, x_n)) = \alpha(f((\gamma \circ \alpha)(x_1), \ldots, (\gamma \circ \alpha)(x_n)))$. Hence the precision of an optimal abstract operator can be reformulated in terms of properties of α, γ and the corresponding concrete operator. The above definitions are naturally extended to "primitive" semantic operators from $\wp(C)$ to C.

[5] There is not presently an agreement on a name for what we call precision. For instance: [61] calls it full-completeness; [37, 80, 86, 90] use the term *completeness*; while [38] use the term *optimality* for the same notion. We prefer to use the term precision, since completeness may be confused with the completeness of a semantics.

Note that if $\bigsqcup$ is the *lub* operation over $(C, \sqsubseteq)$ and $\langle \alpha, \gamma \rangle$ is a Galois insertion then $\widetilde{\bigsqcup} = \alpha \circ \bigsqcup \circ \gamma$ is the *lub* of $(A, \leq)$ and is precise, i.e., $\widetilde{\bigsqcup} \circ \alpha = \alpha \circ \bigsqcup$ (which is equivalent to $\alpha \circ \bigsqcup = \alpha \circ \bigsqcup \circ \gamma \circ \alpha$).

1.5 Quick Reference: Notation, Terminology and Basic Assumptions

We summarize most of the notations used in the thesis (introduced in the previous sections) and the basic assumptions for the experienced reader's convenience.

Basic Set Theory

We write $\emptyset$ for the null or empty set and $\mathbb{N}$ for the set of natural numbers $0, 1, 2, \ldots$. The cardinality of a set S is denoted by $card(S)$. The *powerset* of a set S is denoted by $\wp(S)$. By $\{x_i\}_{i \in I}$ we denote the set of objects x_i indexed by $i \in I$.

By $\exists x, y, \ldots, z.\, P(x, y, \ldots, z)$ we abbreviate $\exists x. \exists y. \cdots \exists z.\, P(x, y, \ldots, z)$ and by $\forall x, y, \ldots, z.\, P(x, y, \ldots, z)$ we abbreviate $\forall x. \forall y. \cdots \forall z.\, P(x, y, \ldots, z)$. Moreover, if $E(x_1, \ldots, x_n)$ is an expression and $P(x_1, \ldots, x_n)$ is a property, by $\{E(x_1, \ldots, x_n) \mid x_1 \in S_1, \ldots, x_n \in S_n, P(x_1, \ldots, x_n)\}$ we abbreviate $\{y \mid \exists x_1 \in S_1, \ldots, x_n \in S_n.\, y = E(x_1, \ldots, x_n), P(x_1, \ldots, x_n)\}$.

By $f : S \rightharpoonup S'$ we denote a partial function of the set S into the set S'. The set of all partial functions from S to S' is denoted by $[S \rightharpoonup S']$. When $f(x)$ is undefined, we write $f(x) = \aleph$, where $\aleph$ denotes the undefined element. For each set S we assume that $\aleph \subseteq S$, $\aleph \cup S = S$ and $\emptyset \not\subseteq \aleph$. Given a partial function $f : S \rightharpoonup S'$, the set $supp(f) := \{x \in S \mid f(x) \text{ is defined}\}$ is the *support* of f. A partial function is said to be *finite-support* if $supp(f)$ is finite. Moreover, it is said to be *finite* if both $supp(f)$ and $\{f(x) \in S' \mid \exists x \in S.\, f(x) \text{ is defined}\}$ are finite. We will denote (by cases) a finite-support function f, which assumes on input values $v_1, \ldots, v_n$ output values $r_1, \ldots, r_n$ and is otherwise undefined, by

$$f := \begin{cases} v_1 \mapsto r_1 \\ \quad \vdots \\ v_n \mapsto r_n \end{cases}$$

Furthermore, if the support of f is just the singleton $\{v\}$, we will denote it by $f := v \mapsto r$.

We use the lambda notation also to denote partial functions by allowing expressions in lambda-terms that are not always defined. Hence, a lambda expression $\lambda x \in S.\, E$ denotes a partial function $S \rightharpoonup S'$ which, on input $x \in S$, assumes the value $E[x] \in S'$, if the expression $E[x]$ is defined, and otherwise it is undefined.

To indicate that a function f from S to S' is total, we write $f : S \to S'$. Moreover, the set of all (total) functions from S to S' is denoted by $[S \to S']$. We denote by $f = g$ the extensional equality. Furthermore, $g := f[^{v}/_{x}]$ denotes the function g which differs from f only for the assignment of v to x.

The *equivalence class* of an element $x \in S$, with respect to $\approx$, is denoted by $[x]_{\approx}$. When clear from the context we abbreviate $[x]_{\approx}$ by $[x]$ and often abuse notation by letting the elements of a set denote their correspondent equivalence classes.

Domain Theory

Let $(S, \leq)$ be a a poset and $X \subseteq S$. A least upper bound of X is denoted by $\mathit{lub}\, X$ or by $\bigsqcup X$. We also write $\bigsqcup\{d_1, \ldots, d_n\}$ as $d_1 \sqcup \cdots \sqcup d_n$. A greatest lower bound of X is denoted by $\mathit{glb}\, X$ or by $\bigsqcap X$. We also write $\bigsqcap\{d_1, \ldots, d_n\}$ as $d_1 \sqcap \cdots \sqcap d_n$. When it is clear from the context, the subscript S will be omitted. Let $\top$ denote the *top element* $\bigsqcup S = \bigsqcap \emptyset$ and $\bot$ denote the *bottom element* $\bigsqcap S = \bigsqcup \emptyset$ of S. Given a complete lattice $(L, \leq)$, the set of all partial functions $F = [S \rightharpoonup L]$ inherits the complete lattice structure of L; $f \preceq g := \forall x \in S.\, f(x) \leq g(x)$, $(f \sqcup g)(x) := f(x) \sqcup g(x)$ and $\bot_F := \lambda x \in S.\, \bot_L$. In the following $f + g$ will be used to denote $f \sqcup g$.

We will use the standard notation $T{\uparrow}_{\alpha} := T{\uparrow}_{\alpha}(\bot)$, where $\bot$ is the least element of S, to denote ordinal powers of an operator $T : S \to S$. In particular, $T{\uparrow}_{\omega} := \bigsqcup_{n<\omega} T{\uparrow}_{n}$, $T{\uparrow}_{n+1} := T(T{\uparrow}_{n})$, for $n < \omega$, and $T{\uparrow}_{0} := \bot$, where $\bigsqcup$ is the *lub* operation of S. Sometimes, $T{\uparrow}_{\alpha}(x)$ may be denoted simply by $T^{\alpha}(x)$.

Logic Programming

In the following, by $\mathbf{t}$ and $\mathbf{x}$ we denote tuples of, respectively, terms and *distinct* variables. Moreover, we denote by $\mathbf{t}$ both the tuple and the set of corresponding syntactic objects. An expression E is any syntactic object, i.e., terms, formulas, *etc.* A *ground* expression is a variable-free expression. In the following the set of variables that occurs in a syntactic object E is denoted by $\mathit{var}(E)$. Moreover, by $\mathit{preds}(E)$ we denote the set of predicates occurring in E.

By ε we denote the *empty substitution* (i.e., the identity function). Let $\mathbf{x} := x_1, \ldots, x_n$ and $\mathbf{t} := t_1, \ldots, t_n$. In the following, if (for any $i \in [1, n]$) $x_i \neq t_i$, then $\{\mathbf{x}/\mathbf{t}\}$ denotes the substitution $\{x_1/t_1, \ldots, x_n/t_n\}$. $\mathit{dom}(\vartheta)$ and $\mathit{range}(\vartheta)$ denote the *domain* and *range* of ϑ. If ϑ is a substitution and $W \subseteq V$ is a set of variables, $\vartheta|_W$ is the restriction of ϑ to the variables W. In the following, when no ambiguity arise, we will denote the substitution $\vartheta|_{\mathit{var}(E)}$ simply by $\vartheta|_E$. A substitution ϑ is *idempotent* if $\vartheta\vartheta = \vartheta$ or, equivalently, if $\mathit{dom}(\vartheta) \cap \mathit{range}(\vartheta) = \emptyset$. In the following, we restrict our attention to idempotent substitutions, unless explicitly stated otherwise. The set of all idempotent substitutions is denoted by *Subst*.

The pre-ordering $\leq$ (more general than) on substitutions is such that $\vartheta \leq \sigma$ if and only if there exists ϑ' such that $\vartheta\vartheta' = \sigma$. Similarly, given two terms t and t', we define $t \leq t'$ (t is more general than t') if and only if there exists a substitution ϑ such that $t\vartheta = t'$. The relation $\leq$ is a preorder and by $\equiv$ we denote the associated equivalence relation (*variance*).

If $\mathbf{t} := t_1, \ldots, t_n$ and $\mathbf{s} := s_1, \ldots, s_n$, a (most general) unifier of $\mathbf{t}$ and $\mathbf{s}$ is a (most general) unifier of $(s_1, \ldots, s_n)$ and $(t_1, \ldots, t_n)$. Since idempotent *mgus* of t_1 and t_2 are unique up to renaming, we will denote by $mgu(t_1, t_2)$ the idempotent most general unifier of t_1 and t_2 obtained by any (arbitrary but once and for all given) choice between the possibly equivalent *mgus*. $\mathbf{t} =_e \mathbf{s}$ denotes the equation set $\{s_1 =_e t_1, \ldots, s_n =_e t_n\}$. If $A := p(\mathbf{s})$ and $B := p(\mathbf{t})$, $A =_e B$ is a shorthand for $\mathbf{s} =_e \mathbf{t}$. This notation is extended to conjunction of atoms in the obvious way. In the following, sets and conjunctions of equations will often be identified and, when clear from the context, we abuse notation denoting $=_e$ by $=$.

Given an equation set $E := \{\mathbf{s} =_e \mathbf{t}\}$, a (most general) unifier of E is a (most general) unifier of $\mathbf{s}$ and $\mathbf{t}$. The preordering $\leq_e$ on equation sets is such that $E \leq_e E'$ if and only if the solutions of E are also solutions of E'. Two sets E, E' are called *equivalent* (denoted by $E \approx_e E'$) if they have the same solutions. If E is the solved form equation set $\{v_1 =_e t_1, \ldots, v_n =_e t_n\}$, the variables v_i are said to be *eliminable*. The set $\{v_1, \ldots, v_n\}$ is denoted by $elim(E)$. In the following, given a set of solved form equation sets $\mathcal{E}$, by $elim(\mathcal{E})$ we denote $\bigcup_{E \in \mathcal{E}} elim(E)$.

Throughout the thesis we will assume logic programs and goals being defined on a fixed first order language given by a non-empty *signature* Σ, a finite set Π of *predicates* and a denumerable set V of *variables*. A (predicate or function) symbol f with arity n will be denoted by f/n. Function symbols whose arity is 0 are called also *constant* symbols. We assume that V, Σ and Π are pairwise disjoint. $T(\Sigma, V)$ denotes the set of *terms* and $T(\Sigma)$ denotes the set of ground terms. The set of all atoms is denoted by *Atoms*. An atom is called *pure* if it is in the form $p(\mathbf{x})$. We will denote a goal by the sequence of atoms $A_1, \ldots, A_n$. If $n = 1$, the goal is called *atomic* and, by an abuse of notation, it is denoted with the corresponding atom. Moreover, if $n = 0$ the goal is called *empty* and is denoted by $\Box$. The set of all goals is denoted by *Goals*. A goal is called *pure* if it contains only pure atoms which do not share variables. We denote by $\mathbf{G}$ and $\mathbf{B}$ possibly empty sequences of atoms. A clause is denoted by $H \leftarrow \mathbf{B}$. By $\mathbf{B}, \mathbf{B}'$ we denote the concatenation of $\mathbf{B}$ and $\mathbf{B}'$. A *query* is the union of a goal $\mathbf{G}$ with a logic program P, here denoted by the formula $\mathbf{G}$ *in* P.

An *equational goal* is an object of the form $(\mathcal{E}, \mathbf{B})$ where $\mathbf{B}$ is a pure goal and $\mathcal{E}$ is a finite set of solved form equation sets such that $elim(\mathcal{E}) \subseteq var(\mathbf{B})$. $\{\emptyset\}, \mathbf{B}$ will be denoted by $\mathbf{B}$. An *equational clause* is a formula of the form $H \leftarrow E, \mathbf{B}$, where $(H, \mathbf{B})$

is a pure goal and E is a solved form equation set such that $\mathit{elim}(\mathsf{E}) \subseteq \mathit{var}(\mathsf{H}, \mathbf{B})$. In the following, given any program clause $\mathsf{p}(\mathbf{t}) \leftarrow \mathsf{p}_1(\mathbf{t}_1), \ldots, \mathsf{p}_n(\mathbf{t}_n)$, we will consider (when needed) its equational form $\mathsf{p}(\mathbf{x}) \leftarrow \mathsf{E}, \mathsf{p}_1(\mathbf{x}_1), \ldots, \mathsf{p}_n(\mathbf{x}_n)$ where $\mathsf{E} := \{\mathbf{x} =_e \mathbf{t}, \mathbf{x}_1 =_e \mathbf{t}_1, \ldots, \mathbf{x}_n =_e \mathbf{t}_n\}$ (and $\mathbf{x}, \mathbf{x}_1, \ldots, \mathbf{x}_n$ are new distinct variables).

The *Herbrand Universe* $\mathcal{U}_\mathsf{L}$ for the language L is the set of ground terms $\mathsf{T}(\Sigma)$, while the *Herbrand Base* $\mathcal{B}_\mathsf{L}$ of L is the set of all ground atoms of L. In the following we will assume that Σ contains at least one function symbol of arity 0 (a constant) and therefore the Herbrand universe is not empty.

Given a derivation $\mathtt{d}$, $\mathit{first}(\mathtt{d})$ and $\mathit{last}(\mathtt{d})$ (if $\mathtt{d}$ is finite) are respectively the first and the last goal of $\mathtt{d}$. By $\mathit{clauses}(\mathtt{d})$ we denote the sequence of clauses of $\mathtt{d}$ and by $\mathit{answer}(\mathtt{d})$ we denote the (partial) computed answer substitution of $\mathtt{d}$. Moreover, $\mathit{length}(\mathtt{d})$ denotes the length of the derivation $\mathtt{d}$, i.e., the number of the *SLD*-derivation steps. By an abuse of notation, we denote a zero-length derivation of $\mathbf{G}$ by $\mathbf{G}$ itself. Finally, $\mathit{prefix}(\mathtt{d})$ is the set of all *SLD*-derivations which are prefixes of $\mathtt{d}$.

In the following, for the sake of simplicity, we will consider the PROLOG selection rule. All our results can be generalized to skeleton rules [54] (the leftmost rule is a particular case of skeleton rules). *SLD*-derivations via the leftmost rule are called *LD-derivations*. In the following $\mathbf{G} \xrightarrow[\mathsf{c}]{\vartheta} \mathbf{B}$ denotes an *LD*-derivation step of $\mathbf{G}$ which uses the input clause c, where ϑ is the computed *mgu* and $\mathbf{B}$ is the resolvent. Moreover, by

$$\mathtt{d} := \mathbf{G} \xrightarrow[\mathsf{c}_1]{\vartheta_1} \cdots \xrightarrow[\mathsf{c}_n]{\vartheta_n} \mathbf{G}_n$$

we denote a (finite) *LD*-derivation of $\mathbf{G}$ *in* P, where any c_i is a renamed clause of P. We also denote by $\mathbf{G} \xrightarrow[\mathsf{P}]{\vartheta}{}^* \mathbf{B}$ a finite *LD*-derivation of $\mathbf{G}$ *in* P, where ϑ is the (partial) computed answer substitution and $\mathbf{B}$ is the last resolvent.

Equational derivations (via the leftmost selection rule) are formally defined as follows. Let us consider the equational goal $\mathsf{E}, \mathsf{A}_1, \ldots, \mathsf{A}_n$ and let $\mathsf{c} := \mathsf{H} \leftarrow \mathbf{B}$ be a renamed apart clause such that $\mathsf{E}' := \mathsf{E} \cup \{\mathsf{A}_1 =_e \mathsf{H}\}$ is unifiable. Then we have an *equational derivation step* $\mathsf{E}, \mathsf{A}_1, \ldots, \mathsf{A}_n \xrightarrow[\mathsf{c}]{\mathrm{eq}} \mathsf{E}', \mathbf{B}, \mathsf{A}_2, \ldots, \mathsf{A}_n$. Equational derivations are obtained from derivation steps in the usual way. A successful equational derivation is a finite one which has only equational atoms in the last resolvent. In the following, to simplify the notation $\mathbf{G}_0 \xrightarrow[\mathsf{c}_1,\ldots,\mathsf{c}_n]{\vartheta} \mathbf{G}_n$ will represent the derivation $\mathtt{d} = \mathbf{G}_0 \xrightarrow[\mathsf{c}_1]{\vartheta_1} \cdots \xrightarrow[\mathsf{c}_n]{\vartheta_n} \mathbf{G}_n$ such that $\vartheta = \vartheta_1 \cdots \vartheta_n$ and $\mathsf{E}_0, \mathbf{G}_0 \xrightarrow[\mathsf{c}_1,\ldots,\mathsf{c}_n]{\mathrm{eq}} \mathsf{E}_n, \mathbf{G}_n$ will represent the equational derivation $\mathsf{E}_0, \mathbf{G}_0 \xrightarrow[\mathsf{c}_1]{\mathrm{eq}} \cdots \xrightarrow[\mathsf{c}_n]{\mathrm{eq}} \mathsf{E}_n, \mathbf{G}_n$.

Abstract Interpretation

Consider a concrete semantics and a Galois insertion between the concrete and the abstract domain. The theory requires the concrete semantics to be the least fixpoint of a semantic function $\mathsf{F} : \mathsf{C} \to \mathsf{C}$. The abstract semantic function $\tilde{\mathsf{F}} : \mathsf{A} \to \mathsf{A}$ is *correct* if $\forall x \in \mathsf{C}.\, \mathsf{F}(x) \sqsubseteq \gamma(\tilde{\mathsf{F}}(\alpha(x)))$. It is often the case (at least in this thesis) that F is in turn defined as composition of "primitive" operators. Let $f : \mathsf{C}^n \to \mathsf{C}$ be one such an operator and assume that $\tilde{f}$ is its abstract counterpart. Then $\tilde{f}$ is *(locally) correct* w.r.t. f if $\forall x_1, \ldots, x_n \in \mathsf{C}.\, f(x_1, \ldots, x_n) \sqsubseteq \gamma(\tilde{f}(\alpha(x_1), \ldots, \alpha(x_n)))$. The abstract operator $\tilde{f}$ is *precise* if $\forall x_1, \ldots, x_n \in \mathsf{C}.\, \alpha(f(x_1, \ldots, x_n)) = \tilde{f}(\alpha(x_1), \ldots, \alpha(x_n))$ which is equivalent to $\alpha(f(x_1, \ldots, x_n)) = \alpha(f((\gamma \circ \alpha)(x_1), \ldots, (\gamma \circ \alpha)(x_n)))$. Hence the precision of an optimal abstract operator can be reformulated in terms of properties of α, γ and the corresponding concrete operator. The above definitions are naturally extended to "primitive" semantic operators from $\wp(\mathsf{C})$ to C.

Part I

A Theory of Observables for Logic Programs

Chapter 2

The Collecting Semantics of the Framework

Lack of compositionality of conventional logic programming semantics has been a serious limitation, since by their very nature PROLOG program fragments are written to be used in an extensible, modular fashion. It has often been noted, in particular, that traditional bottom-up and top-down semantics fail to be sufficiently operational, identify too many computationally distinct programs and are blind to many interesting observables.

The chapter introduces a semantics for definite logic programs, expressed in term of *SLD*-derivations, and studies various properties of *SLD*-derivations by using the above semantics. The semantics is defined according to the approach in [13], which was already used for some abstractions of *SLD*-derivations, such as computed answers [45], call patterns and partial answers [52] and resultants [54]. The basic idea underlying the approach is the *goal independent program denotation*, which can equivalently be specified by top-down and bottom-up constructions. The top-down definition is the set of *SLD*-derivations for pure atomic goals, while the bottom-up definition is the least fixpoint of a suitable immediate consequence operator. The denotation is proved to be correct and minimal w.r.t. the observational equivalence induced on programs by *SLD*-derivations. Moreover, it is proved to enjoy two important compositionality properties, i.e., *AND-compositionality* and *OR-compositionality*. *AND*-compositionality means that the *SLD*-derivations of any goal can be reconstructed from the goal independent denotation. *OR*-compositionality means that the denotation of $\mathsf{P}_1 \cup \mathsf{P}_2$ can be reconstructed from the denotations of P_1 and P_2.

The above results simply extend to *SLD*-derivations similar results obtained for other (more abstract) observables. The main novelty of this work is the semantic definition methodology and the structure of the resulting semantics. We will start by

defining a *denotational semantics* on domains consisting of sets of *SLD*-derivations. It is a rather standard denotational definition with two peculiarities. First it deals with low-level operational details, while the usual denotational semantics operates on the domain of computed answers and is therefore much more abstract. Moreover, the typical compositional style of denotational semantics allows us to identify a small set of *primitive semantic operators*, which are the semantic counterpart of the language syntactic operators. The same primitive semantic operators are then used to define the operational semantics, by means of a *transition system*.

The proof of all the main theorems, such as

- equivalence between denotational and operational semantics,
- equivalence between bottom-up and top-down (goal independent) denotations,
- correctness and minimality of the denotation,
- *AND*-compositionality and *OR*-compositionality of the denotation,

heavily rely on some lemmata, which express properties of the primitive semantic operators. This is even more important, because the *SLD*-derivations semantics has been conceived as the collecting semantics for a hierarchy of semantics (see Chapter 3), systematically derived by using abstract interpretation theory [32]. Its definition style (denotational semantics and transition system) will be inherited by all the abstract semantics. Moreover, since abstraction is essentially abstraction of the primitive semantic operators, the abstract semantics will inherit all those properties of the collecting semantics for which the suitable lemmata on the semantic operators hold. This provides the basis for the definition of a taxonomy of abstractions [21, 22], which is fully presented in Chapter 3. It is worth noting that the *SLD*-derivation semantics is the most natural choice for a collecting semantics. It is essentially a trace semantics and it contains all the relevant information of *SLD*-trees. A more abstract semantics, such as the resultant semantics, would not allow to derive properties such as proof trees (used in the Heyting's semantics in [66, 67]) or derivation lengths.

The chapter is organized as follows. Section 2.1 defines the semantic domain. Section 2.2 introduces the denotational semantics and the primitive semantic operators. Section 2.3 defines the transition system. Section 2.4 defines the goal independent denotations. Section 2.5 contains the main equivalence and compositionality theorems. Finally Section 2.7 contains the proofs of the technical results.

The results of this chapter are from [30, 21].

2.1 Semantic Domain

We assume the reader to be familiar with the notions of *SLD*-resolution and *SLD*-tree introduced in Section 1.3. We want to represent here, for notational convenience, *SLD*-trees as sets of derivations.

1. A set of derivations S is *well-formed* if and only if, for any $\mathtt{d} \in \mathsf{S}$, $\mathit{prefix}(\mathtt{d}) \subseteq \mathsf{S}$. Each well-formed set where all the clauses used in derivations are in P is a representation of a family of partial *SLD*-trees of P.

2. We denote by *WFS* the complete lattice of well-formed sets of derivations, partially ordered by inclusion. The maximal well-formed set of derivations of $\mathbf{G}$ *in* P is a representation of the *SLD*-tree of $\mathbf{G}$ *in* P.

3. A *collection* D is a partial function $\mathit{Goals} \rightharpoonup \mathit{WFS}$ such that, for every $\mathbf{G} \in \mathit{Goals}$, $\mathsf{D}(\mathbf{G})$ is a well-formed set of derivations *all starting from* $\mathbf{G}$ or it is undefined. Formally, $\forall \mathtt{d} \in \mathsf{D}(\mathbf{G}).\ \mathit{first}(\mathtt{d}) = \mathbf{G}$. Hence a collection is a partial function which associates to any goal $\mathbf{G}$ a (representation of) a partial *SLD*-tree of $\mathbf{G}$ *in* P. A *pure* collection is a collection defined only for pure atomic goals.

4. $\mathbb{C}$ is the domain of all the collections ordered the by lifting of inclusion to functions, i.e., $\mathsf{D} \sqsubseteq \mathsf{D}'$ if and only if $\forall \mathbf{G}.\, \mathsf{D}(\mathbf{G}) \subseteq \mathsf{D}'(\mathbf{G})$. The partial order on $\mathbb{C}$ formalizes the evolution of the computation process. It is easy to prove that $(\mathbb{C}, \sqsubseteq)$ is a complete lattice. By $\mathbb{PC}$ we denote the sub-lattice of all pure collections.

Example 2.1.1

Consider the *SLD*-tree of Figure 2.1. The collection D, which represents it, is a function which associates $\mathbf{G}$ to the (well-formed) set of derivations

$$\left\{\mathbf{G};\ \ \mathbf{G} \xrightarrow[c_1]{\vartheta_1} \mathbf{A};\ \ \mathbf{G} \xrightarrow[c_2]{\vartheta_2} \mathbf{B};\ \ \mathbf{G} \xrightarrow[c_3]{\vartheta_3} \mathbf{C};\ \ \mathbf{G} \xrightarrow[c_1]{\vartheta_1} \mathbf{A} \xrightarrow[c_4]{\vartheta_4} \mathbf{D}\right\}$$

and is undefined for all the other inputs.

Figures 2.2 and 2.3 are a representation[1] of the collections

$$\mathsf{D}_1 := \begin{cases} p(x,y) \mapsto \{p(x,y);\ \ d_1;\ \ d_3\} \\ n(x) \quad\ \mapsto \{n(x);\ \ d_5\} \end{cases}$$

[1]In the following examples we will represent by a family of *SLD*-trees, for goals $\mathbf{G}_1, \ldots, \mathbf{G}_n$, the collection which associates to any goal $\mathbf{G}_i$ the well-formed set representing the *SLD*-tree corresponding to $\mathbf{G}_i$ and is undefined for any goal other than $\mathbf{G}_1, \ldots, \mathbf{G}_n$.

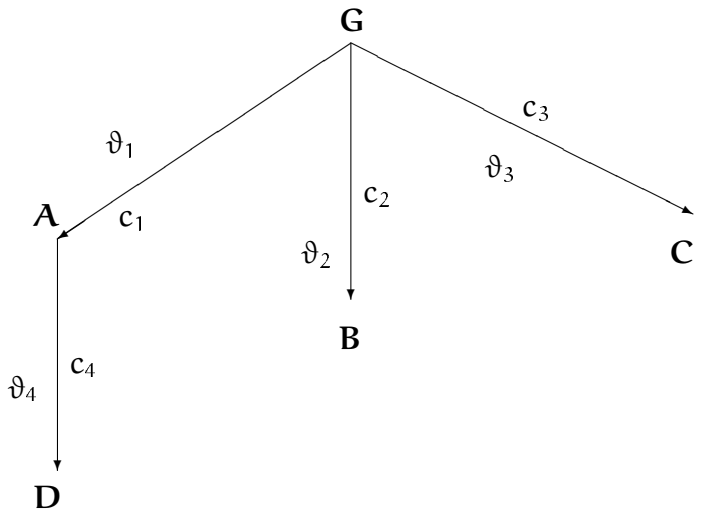

Figure 2.1: The *SLD*-tree corresponding to the collection D in Example 2.1.1

$$D_2 := \begin{cases} q(x,y) & \mapsto \{q(x,y);\ d_7;\ d_2\} \\ q(f(x),z) & \mapsto \{q(f(x),z);\ d_6;\ d_4\} \end{cases}$$

where (as in Example 1.3.7 on page 19)

$$d_1 := p(x,y) \xrightarrow[p(v,w)\leftarrow q(v,w)]{\{v/x,w/y\}} q(x,y)$$

$$d_2 := q(x,y) \xrightarrow[q(t,u)\leftarrow n(f(s),t)]{\{t/x,u/y\}} n(f(s),x) \xrightarrow[n(r,a)\leftarrow]{\{r/f(s),x/a\}} \square$$

$$d_3 := p(x,y) \xrightarrow[p(g(a),b)\leftarrow]{\{x/g(a),y/b\}} \square$$

$$d_4 := q(f(x),z) \xrightarrow[q(t,a)\leftarrow m(g(s),t)]{\{t/f(x),z/a\}} m(g(s),f(x)) \xrightarrow[m(r,f(r))\leftarrow]{\{r/g(s),x/g(s)\}} \square$$

$$d_5 := n(x) \xrightarrow[n(t)\leftarrow p(t,t)]{\{t/x\}} p(x,x)$$

$$d_6 := q(f(x),z) \xrightarrow[q(t,a)\leftarrow m(g(s),t)]{\{t/f(x),z/a\}} m(g(s),f(x))$$

$$d_7 := q(x,y) \xrightarrow[q(t,u)\leftarrow n(f(s),t)]{\{t/x,u/y\}} n(f(s),x)$$

The goal we want to achieve is to develop a denotation modeling partial *SLD*-trees. We follow the approach in [13], by defining a "syntactic" semantic domain (interpretation). Our modeling of partial *SLD*-trees is similar to the basic denotation defined in terms of clauses in [52].

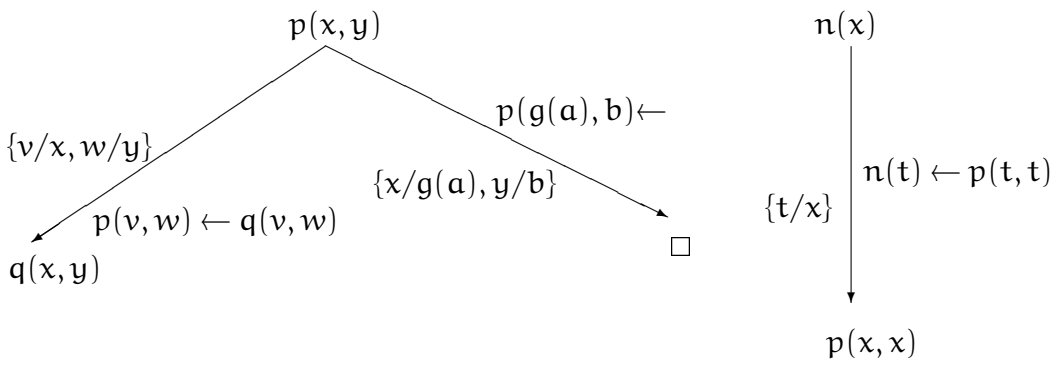

Figure 2.2: The *SLD*-trees corresponding to the collection D_1 in Example 2.1.1

In order for the semantics not to depend upon variable names and on the specific unification algorithm, we define the *equivalence modulo enhanced variance* $\equiv_{\mathbb{C}}$ on collections. Namely, $\mathsf{D} \equiv_{\mathbb{C}} \mathsf{D}'$ if and only if, for any $\mathbf{G}$ such that $\mathsf{D}(\mathbf{G})$ is defined, there exists a variant $\mathbf{G}'$ of $\mathbf{G}$ such that $\mathsf{D}'(\mathbf{G}')$ is defined and, for any $\mathtt{d} \in \mathsf{D}(\mathbf{G})$, there exists $\mathtt{d}' \in \mathsf{D}'(\mathbf{G}')$, such that $\mathit{clauses}(\mathtt{d}) \equiv \mathit{clauses}(\mathtt{d}')$ and vice versa. Hence, derivations with a different choice of the *mgu* and of the new variables introduced by the renaming apart operation are equivalent modulo enhanced variance.

Definition 2.1.2 *An* interpretation I *($\mathbb{C}$-interpretation) is a pure collection modulo enhanced variance. We denote by $\mathbb{I}_{\mathbb{C}}$ the set of interpretations and, by abuse of notation, we denote the induced quotient order on $\mathbb{I}_{\mathbb{C}}$ by $\sqsubseteq$. Thus $(\mathbb{I}_{\mathbb{C}}, \sqsubseteq)$ is a complete lattice.*

Note that in interpretations the enhanced variance relation allows us to abstract w.r.t. the variables names occurring in the initial goals of any collection.

2.2 Denotational Semantics of *SLD*-derivations

Several denotational semantics have been defined for logic programs (see, for example, [64, 65, 39]). The main differences w.r.t. the above definitions are that we do not consider the PROLOG search rule and that our denotations model *SLD*-derivations rather than just computed answers.

We denote the equivalence class (modulo enhanced variance) of a collection σ by σ itself. Moreover, any interpretation I of $\mathbb{I}_{\mathbb{C}}$ is implicitly considered also as an arbitrary collection obtained by choosing an arbitrary representative of I. All the semantic operators that we use on interpretations are independent of the choice of the representative. Therefore, we can define any operator on $\mathbb{I}_{\mathbb{C}}$ in terms of its counterpart defined on $\mathbb{C}$, independently from the choice of the representative. All

$$
\begin{array}{ccc}
q(x,y) & & q(f(x),z) \\
\{t/x,u/y\}\Big\downarrow & q(t,u) \leftarrow n(f(s),t) & \{t/f(x),z/a\}\Big\downarrow \quad q(t,a) \leftarrow m(g(s),t) \\
n(f(s),x) & & m(g(s),f(x)) \\
\{r/f(s),x/a\}\Big\downarrow & n(r,a)\leftarrow & \{r/g(s),x/g(s)\}\Big\downarrow \quad m(r,f(r))\leftarrow \\
\square & & \square
\end{array}
$$

Figure 2.3: The *SLD*-trees corresponding to the collection D_2 in Example 2.1.1

the definitions are independent from the choice of the syntactic object. To simplify the notation, we denote the corresponding operators on $\mathbb{I}_{\mathbb{C}}$ and $\mathbb{C}$ by the same name.

We define the denotational semantics inductively on the syntax of logic programs. Hence, the semantic functions are

$$
\begin{array}{ll}
\mathcal{Q}[\![\cdot]\!] : \mathit{QUERY} \longrightarrow \mathbb{C}, & \\
\mathcal{G}[\![\cdot]\!] : \mathit{GOAL} \longrightarrow (\mathbb{I}_{\mathbb{C}} \to \mathbb{C}), & \mathcal{A}[\![\cdot]\!] : \mathit{ATOM} \longrightarrow (\mathbb{I}_{\mathbb{C}} \to \mathbb{C}), \\
\mathcal{P}[\![\cdot]\!] : \mathit{PROG} \longrightarrow (\mathbb{I}_{\mathbb{C}} \to \mathbb{I}_{\mathbb{C}}), & \mathcal{C}[\![\cdot]\!] : \mathit{CLAUSE} \longrightarrow (\mathbb{I}_{\mathbb{C}} \to \mathbb{I}_{\mathbb{C}})
\end{array}
$$

and are defined in terms of the semantic operators $\cdot$, $\times$, $\triangleright$, $\sum$ defined in Section 2.2.2. The choice of the semantic operators is induced by syntactic operations, due to the compositional nature of definitions in the denotational style. The informal meaning of the operators is the following. The operator $\cdot$ "solves" an atomic goal A into an interpretation I. The operator $\times$ computes the conjunction of two interpretations. The operator $\triangleright$ computes the interpretation obtained by replacement. The operator $\sum$ computes the non-deterministic union of a class of interpretations. Note that when the class is finite we use the infix notation $+$. Finally the function $\phi_\square$ is the collection of the empty goal and *tree* maps clauses to collections.

$$
\begin{aligned}
\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ P]\!] &:= \mathcal{G}[\![\mathbf{G}]\!]_{\mathit{lfp}\,\mathcal{P}[\![P]\!]} && (2.2.1)\\
\mathcal{G}[\![A,\mathbf{G}]\!]_I &:= \mathcal{A}[\![A]\!]_I \times \mathcal{G}[\![\mathbf{G}]\!]_I \qquad \mathcal{G}[\![\square]\!]_I := \phi_\square && (2.2.2)\\
\mathcal{A}[\![A]\!]_I &:= A \cdot I && (2.2.3)\\
\mathcal{P}[\![\{c\}\cup P]\!]_I &:= \mathcal{C}[\![c]\!]_I + \mathcal{P}[\![P]\!]_I \qquad \mathcal{P}[\![\emptyset]\!]_I := \mathit{Id}_{\mathbb{I}} && (2.2.4)\\
\mathcal{C}[\![H \leftarrow \mathbf{B}]\!]_I &:= \mathit{tree}(H \leftarrow \mathbf{B}) \triangleright \mathcal{G}[\![\mathbf{B}]\!]_I, && (2.2.5)
\end{aligned}
$$

where $\mathit{lfp}\,\mathcal{P}[\![P]\!]$ means $\mathit{lfp}_{\mathbb{I}_{\mathbb{C}}}\,\lambda I.\ \mathcal{P}[\![P]\!]_I$.

The last definition (2.2.5) evaluates to the (collection mapping the pure version of the head of the clause to the) one step derivation, using the clause $H \leftarrow \mathbf{B}$ followed

by all (suitably renamed) derivations starting with **B** obtained by composition from I.

We give now a preliminary example to focus the main ideas of the semantics. We refer to Sections 2.2.1 and 2.2.2 for more details.

Example 2.2.1

Consider the (slight modification[2] of the well-known *append*) program P of Figure 2.4 and the goal $\mathbf{G} := \mathsf{ap}([\mathsf{a}],[\mathsf{l}],\mathsf{x}), \mathsf{ap}(\mathsf{x},[\mathsf{h}],\mathsf{z})$. The denotation of $\mathbf{G}$ *in* P is

$$\begin{aligned}\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}]\!] &= \mathcal{G}[\![\mathbf{G}]\!]_{\mathit{lfp}\,\mathcal{P}[\![\mathsf{P}]\!]}\\ &= \mathcal{A}[\![\mathsf{ap}([\mathsf{a}],[\mathsf{l}],\mathsf{x})]\!]_{\mathit{lfp}\,\mathcal{P}[\![\mathsf{P}]\!]} \times \mathcal{A}[\![\mathsf{ap}(\mathsf{x},[\mathsf{h}],\mathsf{z})]\!]_{\mathit{lfp}\,\mathcal{P}[\![\mathsf{P}]\!]} \times \phi_{\square}\\ &= \big(\mathsf{ap}([\mathsf{a}],[\mathsf{l}],\mathsf{x}) \cdot \mathit{lfp}\,\mathcal{P}[\![\mathsf{P}]\!]\big) \times \big(\mathsf{ap}(\mathsf{x},[\mathsf{h}],\mathsf{z}) \cdot \mathit{lfp}\,\mathcal{P}[\![\mathsf{P}]\!]\big)\end{aligned}$$

Since

$$\begin{aligned}\mathcal{P}[\![\mathsf{P}]\!]_{\mathrm{I}} = \mathsf{ap}(\mathsf{x},\mathsf{y},\mathsf{z}) \mapsto \big\{&\mathsf{ap}(\mathsf{x},\mathsf{y},\mathsf{z});\ \ \mathsf{ap}(\mathsf{x},\mathsf{y},\mathsf{z}) \xrightarrow[\mathsf{ap}([\,],\mathsf{v},\mathsf{v})\leftarrow]{\{\mathsf{x}/[\,],\mathsf{y}/\mathsf{v},\mathsf{z}/\mathsf{v}\}} \square;\\ &\mathsf{ap}(\mathsf{x},\mathsf{y},\mathsf{z}) \xrightarrow[\mathsf{ap}([\mathsf{l}|\mathsf{u}],\mathsf{t},[\mathsf{l}|\mathsf{v}])\leftarrow \mathsf{ap}(\mathsf{u},\mathsf{t},\mathsf{v})]{\{\mathsf{x}/[\mathsf{l}|\mathsf{u}],\mathsf{y}/\mathsf{t},\mathsf{z}/[\mathsf{l}|\mathsf{v}]\}} \mathsf{ap}(\mathsf{u},\mathsf{t},\mathsf{v})\big\} \triangleright (\mathsf{ap}(\mathsf{u},\mathsf{t},\mathsf{v}) \cdot \mathrm{I})\end{aligned}$$

and

$$\begin{aligned}\mathit{lfp}\,\mathcal{P}[\![\mathsf{P}]\!] = \mathsf{ap}(\mathsf{x},\mathsf{y},\mathsf{z}) \mapsto \big\{&\mathsf{ap}(\mathsf{x},\mathsf{y},\mathsf{z});\ \ \mathsf{ap}(\mathsf{x},\mathsf{y},\mathsf{z}) \xrightarrow[\mathsf{ap}([\,],\mathsf{v},\mathsf{v})\leftarrow]{\{\mathsf{x}/[\,],\mathsf{y}/\mathsf{v},\mathsf{z}/\mathsf{v}\}} \square;\\ &\mathsf{ap}(\mathsf{x},\mathsf{y},\mathsf{z}) \xrightarrow[\mathsf{ap}([\mathsf{l}|\mathsf{u}],\mathsf{t},[\mathsf{l}|\mathsf{v}])\leftarrow \mathsf{ap}(\mathsf{u},\mathsf{t},\mathsf{v})]{\{\mathsf{x}/[\mathsf{l}|\mathsf{u}],\mathsf{y}/\mathsf{t},\mathsf{z}/[\mathsf{l}|\mathsf{v}]\}} \mathsf{ap}(\mathsf{u},\mathsf{t},\mathsf{v});\\ &\mathsf{ap}(\mathsf{x},\mathsf{y},\mathsf{z}) \xrightarrow[\mathsf{ap}([\mathsf{l}|\mathsf{u}],\mathsf{t},[\mathsf{l}|\mathsf{v}])\leftarrow \mathsf{ap}(\mathsf{u},\mathsf{t},\mathsf{v})]{\{\mathsf{x}/[\mathsf{l}|\mathsf{u}],\mathsf{y}/\mathsf{t},\mathsf{z}/[\mathsf{l}|\mathsf{v}]\}} \mathsf{ap}(\mathsf{u},\mathsf{t},\mathsf{v}) \xrightarrow[\mathsf{ap}([\,],\mathsf{w},\mathsf{w})\leftarrow]{\{\mathsf{u}/[\,],\mathsf{t}/\mathsf{w},\mathsf{v}/\mathsf{w}\}} \square;\\ &\ldots\big\},\end{aligned}$$

then

$$\begin{aligned}\mathsf{ap}([\mathsf{a}],[\mathsf{l}],\mathsf{x}) \cdot \mathit{lfp}\,\mathcal{P}[\![\mathsf{P}]\!] = \mathsf{ap}([\mathsf{a}],[\mathsf{l}],\mathsf{x}) \mapsto \mathit{prefix}\,\big(&\mathsf{ap}([\mathsf{a}],[\mathsf{l}],\mathsf{x})\\ &\xrightarrow[\mathsf{ap}([\mathsf{r}|\mathsf{y}],\mathsf{v},[\mathsf{r}|\mathsf{w}])\leftarrow \mathsf{ap}(\mathsf{y},\mathsf{v},\mathsf{w})]{\{\mathsf{x}/[\mathsf{a}|\mathsf{w}],\mathsf{v}/[\mathsf{l}],\mathsf{y}/[\,],\mathsf{r}/\mathsf{a}\}} \mathsf{ap}([\,],[\mathsf{l}],\mathsf{w}) \xrightarrow[\mathsf{ap}([\,],\mathsf{t},\mathsf{t})\leftarrow]{\{\mathsf{w}/[\mathsf{l}],\mathsf{t}/[\mathsf{l}]\}} \square\big)\\ \mathsf{ap}(\mathsf{x},[\mathsf{h}],\mathsf{z}) \cdot \mathit{lfp}\,\mathcal{P}[\![\mathsf{P}]\!] = \mathsf{ap}(\mathsf{x},[\mathsf{h}],\mathsf{z})) \mapsto \mathit{prefix}\,\big(&\mathsf{ap}(\mathsf{x},[\mathsf{h}],\mathsf{z}) \xrightarrow[\mathsf{ap}([\,],\mathsf{y},\mathsf{y})\leftarrow]{\{\mathsf{x}/[\,],\mathsf{z}/[\mathsf{h}],\mathsf{y}/[\mathsf{h}]\}}\\ &\square\,\big) \cup \mathit{prefix}\,\big(\mathsf{ap}(\mathsf{x},[\mathsf{h}],\mathsf{z}) \xrightarrow[\mathsf{ap}([\mathsf{l}|\mathsf{y}],\mathsf{t},[\mathsf{l}|\mathsf{v}])\leftarrow \mathsf{ap}(\mathsf{y},\mathsf{t},\mathsf{v})]{\{\mathsf{x}/[\mathsf{l}|\mathsf{y}],\mathsf{t}/[\mathsf{h}],\mathsf{z}/[\mathsf{l}|\mathsf{v}]\}}\\ &\mathsf{ap}(\mathsf{y},[\mathsf{h}],\mathsf{v}) \xrightarrow[\mathsf{ap}([\,],\mathsf{u},\mathsf{u})\leftarrow]{\{\mathsf{y}/[\,],\mathsf{u}/[\mathsf{h}],\mathsf{v}/[\mathsf{h}]\}} \square\big) \cup \ldots\end{aligned}$$

[2] I shortened all predicates names to limit the size of the formulas.

```
c1: ap([], Xs, Xs).
c2: ap([X|Xs], Ys, [X|Zs]) :- ap(Xs, Ys, Zs).
```

Figure 2.4: The append program

Thus, the semantics of $\mathbf{G}$ *in* P is

$$\begin{aligned}\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}]\!] = \mathbf{G} \mapsto \mathit{prefix}\ \big(&\mathsf{ap}([\mathsf{a}],[\mathsf{l}],\mathsf{x}), \mathsf{ap}(\mathsf{x},[\mathsf{h}],\mathsf{z}) \xrightarrow[\mathsf{ap}([\mathsf{r}|\mathsf{y}],\mathsf{v},[\mathsf{r}|\mathsf{w}])\leftarrow \mathsf{ap}(\mathsf{y},\mathsf{v},\mathsf{w})]{\{\mathsf{x}/[\mathsf{a}|\mathsf{w}],\mathsf{v}/[\mathsf{l}],\mathsf{y}/[],\mathsf{r}/\mathsf{a}\}} \\ &\mathsf{ap}([],[\mathsf{l}],\mathsf{w}), \mathsf{ap}([\mathsf{a}|\mathsf{w}],[\mathsf{h}],\mathsf{z}) \xrightarrow[\mathsf{ap}([],\mathsf{t},\mathsf{t})\leftarrow]{\{\mathsf{w}/[\mathsf{l}],\mathsf{t}/[\mathsf{l}]\}} \\ &\mathsf{ap}([\mathsf{a},\mathsf{l}],[\mathsf{h}],\mathsf{z}) \xrightarrow[\mathsf{ap}([\mathsf{o}|\mathsf{v}'],\mathsf{u},[\mathsf{o}|\mathsf{s}])\leftarrow \mathsf{ap}(\mathsf{v}',\mathsf{u},\mathsf{s})]{\{\mathsf{o}/\mathsf{a},\mathsf{v}'/[\mathsf{l}],\mathsf{u}/[\mathsf{h}],\mathsf{z}/[\mathsf{a}|\mathsf{s}]\}} \\ &\mathsf{ap}([\mathsf{l}],[\mathsf{h}],\mathsf{s}) \xrightarrow[\mathsf{ap}([\mathsf{o}'|\mathsf{v}''],\mathsf{u}',[\mathsf{o}'|\mathsf{s}'])\leftarrow \mathsf{ap}(\mathsf{v}'',\mathsf{u}',\mathsf{s}')]{\{\mathsf{o}'/\mathsf{l},\mathsf{v}''/[],\mathsf{u}'/[\mathsf{h}],\mathsf{s}/[\mathsf{l}|\mathsf{s}']\}} \\ &\mathsf{ap}([],[\mathsf{h}],\mathsf{s}') \xrightarrow[\mathsf{ap}([],\mathsf{t}',\mathsf{t}')\leftarrow]{\{\mathsf{t}'/[\mathsf{h}],\mathsf{s}'/[\mathsf{h}]\}} \square\big).\end{aligned}$$

See Figures 2.5 and 2.6 for an *SLD*-tree representation.

2.2.1 Basic Operators on Derivations

In order to define the semantic operators we first need to define three auxiliary operations on derivations. Such operators will be used later to define the semantic operations on interpretations. The first operation formalizes the concatenation of two derivations, the second computes the instantiation of a derivation, and finally the third defines the *AND*-compositional conjunction of two derivations. These are the formal definitions.

1. Let d_1, d_2 be derivations such that $\mathit{last}(\mathsf{d}_1) = \mathit{first}(\mathsf{d}_2)$ and $\mathit{var}(\mathsf{d}_1) \cap \mathit{var}(\mathsf{d}_2) = \mathit{var}(\mathit{first}(\mathsf{d}_2))$. Then $\mathsf{d}_1 :: \mathsf{d}_2$ denotes the concatenation of d_1 and d_2.

2. Let $\mathsf{d} := \mathbf{G}'_0 \xrightarrow[c_1]{\vartheta'_1} \cdots \xrightarrow[c_k]{\vartheta'_k} \mathbf{G}'_k$ be a derivation and δ be an idempotent substitution such that $\mathit{var}(\mathbf{G}'_0\delta) \cap \mathit{var}(\mathit{clauses}(\mathsf{d})) = \emptyset$. Then $\partial_\delta(\mathsf{d}) := \mathbf{G}_0 \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_h]{\vartheta_h} \mathbf{G}_h$ where

   - $\mathbf{G}_0 := \mathbf{G}'_0\delta$ and
   - for any $0 < i \leq k$, if $\mathbf{G}_{i-1} = (A, \bar{\mathbf{G}}_i)$ and $c_i = H \leftarrow \mathbf{B}$ then (if an *mgu* exists) $\vartheta_i := \mathit{mgu}(A, H)$ and $\mathbf{G}_i := (\mathbf{B}, \bar{\mathbf{G}}_i)\vartheta_i$.

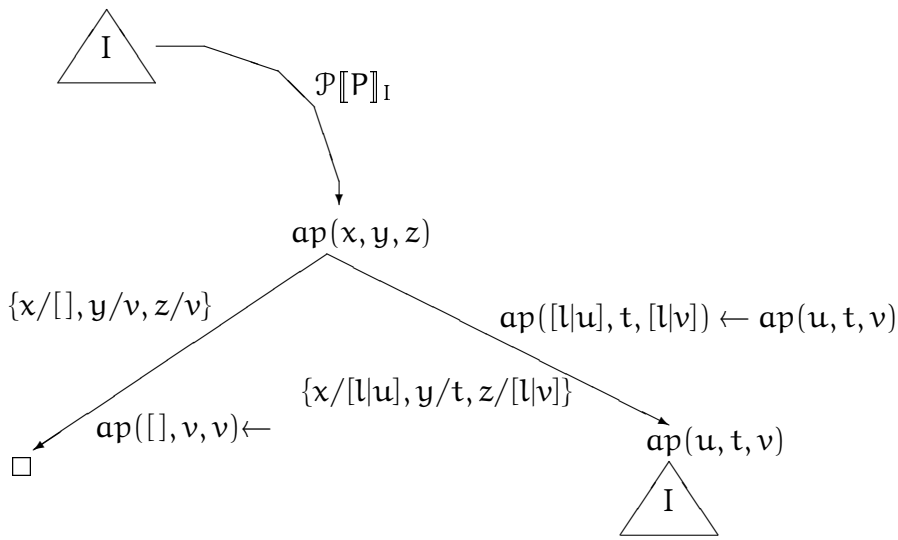

Figure 2.5: The $\mathcal{P}[\![P]\!]$ operator of Example 2.2.1

Note that $\partial_\delta(\mathtt{d})$ is the derivation obtained by applying the substitution δ to *first*$(\mathtt{d})$ and attempting to build as long a derivation as possible (until a failure in finding an *mgu* occurs) using the same clauses as in $\mathtt{d}$. Thus, in particular, $h \leq k$.

3. Let $\mathtt{d}_1 := \mathbf{G}'_0 \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \mathbf{G}'_k$, $\mathtt{d}_2$ be derivations such that $\mathbf{G}''_0 = \mathit{first}(\mathtt{d}_2)$ and $\mathit{var}(\mathtt{d}_1) \cap \mathit{var}(\mathtt{d}_2) = \mathit{var}(\mathbf{G}'_0) \cap \mathit{var}(\mathbf{G}''_0)$. Then

$$\mathtt{d}_1 \wedge \mathtt{d}_2 := \begin{cases} (\mathbf{G}'_0, \mathbf{G}''_0) \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} (\mathbf{G}'_k, \mathbf{G}''_0\vartheta_1 \cdots \vartheta_k) & \text{if } \mathbf{G}'_k \neq \square \\ (\mathbf{G}'_0, \mathbf{G}''_0) \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \mathbf{G}''_0\vartheta_1 \cdots \vartheta_k :: \partial_{\vartheta_1 \cdots \vartheta_k}(\mathtt{d}_2) & \text{otherwise} \end{cases}$$

Note that $\mathtt{d}_1 \wedge \mathtt{d}_2$ is the derivation obtained by adding (a suitable instantiation of) the goal *first*$(\mathtt{d}_2)$ to each goal in $\mathtt{d}_1$ and then (if $\mathtt{d}_1$ is a refutation) attempting to build as long a derivation as possible using the same clauses as in $\mathtt{d}_2$.

The constraints on the variables of derivations are used to avoid variable name clashes in the clauses. Moreover, note that, for any choice of the *mgu* used in the construction of the derivations, the results are equivalent modulo variance.

Example 2.2.2

Consider (as in Example 2.1.1) the following derivations.

$$\mathtt{d}_1 := p(x, y) \xrightarrow[p(v,w) \leftarrow q(v,w)]{\{v/x, w/y\}} q(x, y)$$

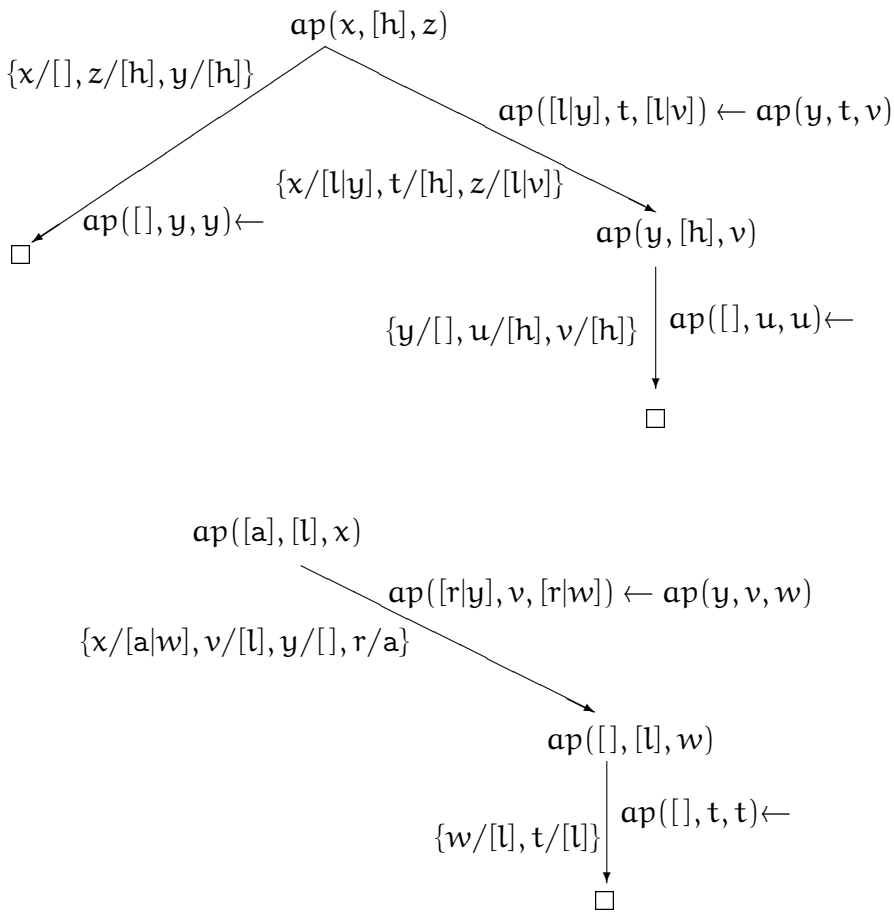

Figure 2.6: The *SLD*-trees of Example 2.2.1

$$\mathtt{d}_2 := \mathtt{q}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{q}(\mathtt{t},\mathtt{u})\leftarrow\mathtt{n}(\mathtt{f}(\mathtt{s}),\mathtt{t})]{\{\mathtt{t}/\mathtt{x},\mathtt{u}/\mathtt{y}\}} \mathtt{n}(\mathtt{f}(\mathtt{s}),\mathtt{x}) \xrightarrow[\mathtt{n}(\mathtt{r},\mathtt{a})\leftarrow]{\{\mathtt{r}/\mathtt{f}(\mathtt{s}),\mathtt{x}/\mathtt{a}\}} \Box$$

$$\mathtt{d}_3 := \mathtt{p}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{p}(\mathtt{g}(\mathtt{a}),\mathtt{b})\leftarrow]{\{\mathtt{x}/\mathtt{g}(\mathtt{a}),\mathtt{y}/\mathtt{b}\}} \Box$$

$$\mathtt{d}_4 := \mathtt{q}(\mathtt{f}(\mathtt{x}),\mathtt{z}) \xrightarrow[\mathtt{q}(\mathtt{t},\mathtt{a})\leftarrow\mathtt{m}(\mathtt{g}(\mathtt{s}),\mathtt{t})]{\{\mathtt{t}/\mathtt{f}(\mathtt{x}),\mathtt{z}/\mathtt{a}\}} \mathtt{m}(\mathtt{g}(\mathtt{s}),\mathtt{f}(\mathtt{x})) \xrightarrow[\mathtt{m}(\mathtt{r},\mathtt{f}(\mathtt{r}))\leftarrow]{\{\mathtt{r}/\mathtt{g}(\mathtt{s}),\mathtt{x}/\mathtt{g}(\mathtt{s})\}} \Box$$

Since $var(\mathtt{d}_1) \cap var(\mathtt{d}_2) = var(\mathtt{d}_3) \cap var(\mathtt{d}_2) = \{\mathtt{x},\mathtt{y}\}$ and $var(\mathtt{d}_3) \cap var(\mathtt{d}_4) = \{\mathtt{x}\}$, the operations $\mathtt{d}_1 :: \mathtt{d}_2$, $\mathtt{d}_1 \wedge \mathtt{d}_2$, $\mathtt{d}_3 \wedge \mathtt{d}_2$ and $\mathtt{d}_3 \wedge \mathtt{d}_4$ are defined and, in particular, the following holds.

$$\mathtt{d}_1 :: \mathtt{d}_2 = \mathtt{p}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{p}(\mathtt{v},\mathtt{w})\leftarrow\mathtt{q}(\mathtt{v},\mathtt{w})]{\{\mathtt{v}/\mathtt{x},\mathtt{w}/\mathtt{y}\}} \mathtt{q}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{q}(\mathtt{t},\mathtt{u})\leftarrow\mathtt{n}(\mathtt{f}(\mathtt{s}),\mathtt{t})]{\{\mathtt{t}/\mathtt{x},\mathtt{u}/\mathtt{y}\}} \mathtt{n}(\mathtt{f}(\mathtt{s}),\mathtt{x}) \xrightarrow[\mathtt{n}(\mathtt{r},\mathtt{a})\leftarrow]{\{\mathtt{r}/\mathtt{f}(\mathtt{s}),\mathtt{x}/\mathtt{a}\}} \Box$$

$$\mathtt{d}_1 \wedge \mathtt{d}_2 = \mathtt{p}(\mathtt{x},\mathtt{y}),\mathtt{q}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{p}(\mathtt{v},\mathtt{w})\leftarrow\mathtt{q}(\mathtt{v},\mathtt{w})]{\{\mathtt{v}/\mathtt{x},\mathtt{w}/\mathtt{y}\}} \mathtt{q}(\mathtt{x},\mathtt{y}),\mathtt{q}(\mathtt{x},\mathtt{y})$$

$$\mathtt{d}_3 \wedge \mathtt{d}_2 = \mathtt{p}(\mathtt{x},\mathtt{y}),\mathtt{q}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{p}(\mathtt{g}(\mathtt{a}),\mathtt{b})\leftarrow]{\{\mathtt{x}/\mathtt{g}(\mathtt{a}),\mathtt{y}/\mathtt{b}\}} \mathtt{q}(\mathtt{g}(\mathtt{a}),\mathtt{b}) \xrightarrow[\mathtt{q}(\mathtt{t},\mathtt{u})\leftarrow\mathtt{n}(\mathtt{f}(\mathtt{s}),\mathtt{t})]{\{\mathtt{t}/\mathtt{g}(\mathtt{a}),\mathtt{u}/\mathtt{b}\}} \mathtt{n}(\mathtt{f}(\mathtt{s}),\mathtt{g}(\mathtt{a}))$$

$$\mathtt{d}_3 \wedge \mathtt{d}_4 = \mathtt{p}(\mathtt{x},\mathtt{y}),\mathtt{q}(\mathtt{f}(\mathtt{x}),\mathtt{z}) \xrightarrow[\mathtt{p}(\mathtt{g}(\mathtt{a}),\mathtt{b})\leftarrow]{\{\mathtt{x}/\mathtt{g}(\mathtt{a}),\mathtt{y}/\mathtt{b}\}} \mathtt{q}(\mathtt{f}(\mathtt{g}(\mathtt{a})),\mathtt{z}) \xrightarrow[\mathtt{q}(\mathtt{t},\mathtt{a})\leftarrow\mathtt{m}(\mathtt{g}(\mathtt{s}),\mathtt{t})]{\{\mathtt{t}/\mathtt{f}(\mathtt{g}(\mathtt{a})),\mathtt{z}/\mathtt{a}\}}$$
$$\mathtt{m}(\mathtt{g}(\mathtt{s}),\mathtt{f}(\mathtt{g}(\mathtt{a}))) \xrightarrow[\mathtt{m}(\mathtt{r},\mathtt{f}(\mathtt{r}))\leftarrow]{\{\mathtt{r}/\mathtt{g}(\mathtt{a}),\mathtt{s}/\mathtt{a}\}} \Box$$

Moreover, considered that $var(clauses(\mathtt{d}_1)) = \{\mathtt{v},\mathtt{w}\}$, $var(clauses(\mathtt{d}_2)) = \{\mathtt{t},\mathtt{u},\mathtt{s},\mathtt{r}\}$, $var(clauses(\mathtt{d}_3)) = \emptyset$ and $var(clauses(\mathtt{d}_4)) = \{\mathtt{t},\mathtt{s},\mathtt{r}\}$, the operations $\partial_{\{\mathtt{x}/\mathtt{f}(\mathtt{t}),\mathtt{w}/\mathtt{c}\}}(\mathtt{d}_1)$, $\partial_{\{\mathtt{x}/\mathtt{f}(\mathtt{g}(\mathtt{z})),\mathtt{y}/\mathtt{z}\}}(\mathtt{d}_2)$, $\partial_{\{\mathtt{x}/\mathtt{b}\}}(\mathtt{d}_3)$ and $\partial_{\{\mathtt{x}/\mathtt{g}(\mathtt{z})\}}(\mathtt{d}_4)$ are defined and, in particular, the following holds.

$$\partial_{\{\mathtt{x}/\mathtt{f}(\mathtt{t}),\mathtt{w}/\mathtt{c}\}}(\mathtt{d}_1) = \mathtt{p}(\mathtt{f}(\mathtt{t}),\mathtt{y}) \xrightarrow[\mathtt{p}(\mathtt{v},\mathtt{w})\leftarrow\mathtt{q}(\mathtt{v},\mathtt{w})]{\{\mathtt{v}/\mathtt{f}(\mathtt{t}),\mathtt{w}/\mathtt{y}\}} \mathtt{q}(\mathtt{f}(\mathtt{t}),\mathtt{y})$$

$$\partial_{\{\mathtt{x}/\mathtt{f}(\mathtt{g}(\mathtt{z})),\mathtt{y}/\mathtt{z}\}}(\mathtt{d}_2) = \mathtt{q}(\mathtt{f}(\mathtt{g}(\mathtt{z})),\mathtt{z}) \xrightarrow[\mathtt{q}(\mathtt{t},\mathtt{u})\leftarrow\mathtt{n}(\mathtt{f}(\mathtt{s}),\mathtt{t})]{\{\mathtt{t}/\mathtt{f}(\mathtt{g}(\mathtt{z})),\mathtt{u}/\mathtt{z}\}} \mathtt{n}(\mathtt{f}(\mathtt{s}),\mathtt{f}(\mathtt{g}(\mathtt{z})))$$

$$\partial_{\{\mathtt{x}/\mathtt{g}(\mathtt{a}),\mathtt{y}/\mathtt{b}\}}(\mathtt{d}_3) = \mathtt{p}(\mathtt{g}(\mathtt{a}),\mathtt{b}) \xrightarrow[\mathtt{p}(\mathtt{g}(\mathtt{a}),\mathtt{b})\leftarrow]{\varepsilon} \Box$$

$$\partial_{\{\mathtt{x}/\mathtt{g}(\mathtt{z})\}}(\mathtt{d}_4) = \mathtt{q}(\mathtt{f}(\mathtt{g}(\mathtt{z})),\mathtt{z}) \xrightarrow[\mathtt{q}(\mathtt{t},\mathtt{a})\leftarrow\mathtt{m}(\mathtt{g}(\mathtt{s}),\mathtt{t})]{\{\mathtt{t}/\mathtt{f}(\mathtt{g}(\mathtt{a})),\mathtt{z}/\mathtt{a}\}} \mathtt{m}(\mathtt{g}(\mathtt{s}),\mathtt{f}(\mathtt{g}(\mathtt{a})))$$
$$\xrightarrow[\mathtt{m}(\mathtt{r},\mathtt{f}(\mathtt{r}))\leftarrow]{\{\mathtt{r}/\mathtt{g}(\mathtt{a}),\mathtt{s}/\mathtt{a}\}} \Box$$

Note that, since the binding $\mathtt{w}/\mathtt{c}$ does not affect the goal $\mathtt{p}(\mathtt{f}(\mathtt{t}),\mathtt{y})$, head of the derivation $\mathtt{d}_1$, then $\partial_{\{\mathtt{x}/\mathtt{f}(\mathtt{t}),\mathtt{w}/\mathtt{c}\}}(\mathtt{d}_1) = \partial_{\{\mathtt{x}/\mathtt{f}(\mathtt{t})\}}(\mathtt{d}_1)$.

The following lemma states that the above operations are well defined.

Lemma 2.2.3 *Let* $\mathtt{d}_1, \mathtt{d}_2$ *be derivations and* δ *be an idempotent substitution. Then the following properties hold.*

1. *If* $\mathtt{d}_1 :: \mathtt{d}_2$ *is defined then* $\mathtt{d}_1 :: \mathtt{d}_2$ *is a derivation.*
2. *If* $\partial_\delta(\mathtt{d}_1)$ *is defined, then* $\partial_\delta(\mathtt{d}_1)$ *is a derivation.*
3. *If* $\mathtt{d}_1 \wedge \mathtt{d}_2$ *is defined then* $\mathtt{d}_1 \wedge \mathtt{d}_2$ *is a derivation.*

Lemma 2.2.4 *Let* $\mathtt{d}_1, \mathtt{d}_2, \mathtt{d}_3$ *be derivations and* δ *be an idempotent substitution. Then the following holds.*

1. *If* $\partial_\delta(\mathtt{d}_1 \wedge \mathtt{d}_2)$ *and* $\partial_\delta(\mathtt{d}_1) \wedge \partial_\delta(\mathtt{d}_2)$ *are defined then* $\partial_\delta(\mathtt{d}_1 \wedge \mathtt{d}_2) = \partial_\delta(\mathtt{d}_1) \wedge \partial_\delta(\mathtt{d}_2)$.
2. *If* $(\mathtt{d}_1 \wedge \mathtt{d}_2) \wedge \mathtt{d}_3$ *and* $\mathtt{d}_1 \wedge (\mathtt{d}_2 \wedge \mathtt{d}_3)$ *are defined then* $(\mathtt{d}_1 \wedge \mathtt{d}_2) \wedge \mathtt{d}_3 = \mathtt{d}_1 \wedge (\mathtt{d}_2 \wedge \mathtt{d}_3)$.

2.2.2 Basic Operators on Collections

Let $\mathsf{D}, \mathsf{D}_1, \mathsf{D}_2$ be collections in $\mathbb{C}$, $\mathbf{G}$ be a goal and A be an atom.

- The *void collection* ϕ is the collection $\lambda\mathbf{G}.\aleph$, i.e., the *always* undefined function.
- The *identity collection* $Id_{\mathbb{C}}$ is the collection of zero-length derivations for each goal, i.e., $\lambda\mathbf{G}.\{\mathbf{G}\}$, while the *pure* identity collection $Id_{\mathbb{I}}$ is the collection[3] $\lambda\mathsf{p}(\mathbf{x}).\{\mathsf{p}(\mathbf{x})\}$. Moreover, $\phi_{\mathbf{G}}$ denotes the collection $\phi\left[{}^{\{\mathbf{G}\}}/_{\mathbf{G}}\right]$[4].
- The *instantiation* of D with A is

$$\mathsf{A} \cdot \mathsf{D} := \phi\left[{}^{\mathsf{S}}/_{\mathsf{A}}\right] \text{ where } \mathsf{S} := \big\{\partial_\delta(\mathtt{d}) \mid \mathsf{S}' \text{ is a renamed apart (from } \mathsf{A}) \text{ version of } \mathsf{D}(\mathsf{A}'), \text{ for some } \mathsf{A}' \leq \mathsf{A},\ \mathtt{d} \in \mathsf{S}' \text{ and there exists } \delta \text{ such that } \mathsf{A} = \mathit{first}(\mathtt{d})\delta\big\}.$$

- The *product* of D_1 and D_2 is

$$\mathsf{D}_1 \times \mathsf{D}_2 := \lambda\mathbf{G}.\big\{\mathtt{d}_1 \wedge \mathtt{d}_2 \mid (\mathbf{G}_1, \mathbf{G}_2) = \mathbf{G} \text{ and for } i = 1, 2,\ \mathbf{G}'_i \equiv \mathbf{G}_i,\ \mathtt{d}_i \text{ is a renamed version of an element in } \mathsf{D}_i(\mathbf{G}'_i), \text{ such that } \mathbf{G}_i = \mathit{first}(\mathtt{d}_i) \text{ and } \mathtt{d}_1 \wedge \mathtt{d}_2 \text{ is defined}\ \big\}.$$

[3]Note that when we write $\lambda\mathsf{p}(\mathbf{x}).\mathsf{E}$ we denote a partial function which is defined only for inputs of the form $\mathsf{p}(\mathbf{x})$ and is otherwise undefined.

[4]Note that the void collection, viewed as a set of ordered pairs, is just the empty set and $\phi_{\mathbf{G}}$ is just the singleton $\{\langle\mathbf{G}, \{\mathbf{G}\}\rangle\}$.

- The *(compatible) extension* of D_1 by D_2 is

$$D_1 \triangleright D_2 := \lambda \mathbf{G}.\, D_1(\mathbf{G}) \cup \big\{ d_1 :: d_2 \mid d_1 \in D_1(\mathbf{G}),\ \mathbf{G}_2 \equiv \mathit{last}(d_1) \text{ and } d_2 \text{ is a renamed version of an element in } D_2(\mathbf{G}_2), \text{ such that } d_1 :: d_2 \text{ is defined} \big\}.$$

  The $\triangleright$ operator is extensive on the first argument, i.e., $D_1 \sqsubseteq D_1 \triangleright D_2$.

- The *sum* of a class $\{D_j\}_{j\in J}$ is

$$\sum \{D_j\}_{j\in J} := \lambda \mathbf{G}.\, \bigcup_{j\in J} D_j(\mathbf{G}).$$

  and $D_1 + D_2$ denotes[5] $\sum\{D_1, D_2\}$. Note that $D_1 \sqsubseteq D_2 \iff D_1 + D_2 = D_2$ and that the *lub* operation on $(\mathbb{C}, \sqsubseteq)$ coincides with $\sum$.

- The *operator tree* maps clauses to collections. Indeed, every clause $c := p(\mathbf{t}) \leftarrow \mathbf{B}$ can be viewed as the "one step" interpretation (collection)

$$\mathit{tree}(c) := \phi \left[\left\{ p(\mathbf{x});\ p(\mathbf{x}) \xrightarrow[c]{\{\mathbf{x}/\mathbf{t}\}} \mathbf{B} \right\} \Big/ p(\mathbf{x}) \right],$$

  where $\mathbf{x}$ is a tuple of new distinct variables. Moreover, *tree* can be extended to programs simply as $\mathit{tree}(P) := \sum\{\mathit{tree}(c)\}_{c\in P}$.

Example 2.2.5

Consider (as in Example 2.1.1 on page 39) the following collections.

$$D_1 := \begin{cases} p(x,y) \mapsto \{p(x,y);\ d_1;\ d_3\} \\ n(x) \quad\ \mapsto \{n(x);\ d_5\} \end{cases}$$

$$D_2 := \begin{cases} q(x,y) \quad\ \mapsto \{q(x,y);\ d_7;\ d_2\} \\ q(f(x),z) \mapsto \{q(f(x),z);\ d_6;\ d_4\} \end{cases}$$

where

$$d_1 := p(x,y) \xrightarrow[p(v,w)\leftarrow q(v,w)]{\{v/x, w/y\}} q(x,y)$$

$$d_2 := q(x,y) \xrightarrow[q(t,u)\leftarrow n(f(s),t)]{\{t/x, u/y\}} n(f(s),x) \xrightarrow[n(r,a)\leftarrow]{\{r/f(s), x/a\}} \square$$

$$d_3 := p(x,y) \xrightarrow[p(g(a),b)\leftarrow]{\{x/g(a), y/b\}} \square$$

[5] Remember that $+$ is an infix notation for $\sum$ when applied to finite classes.

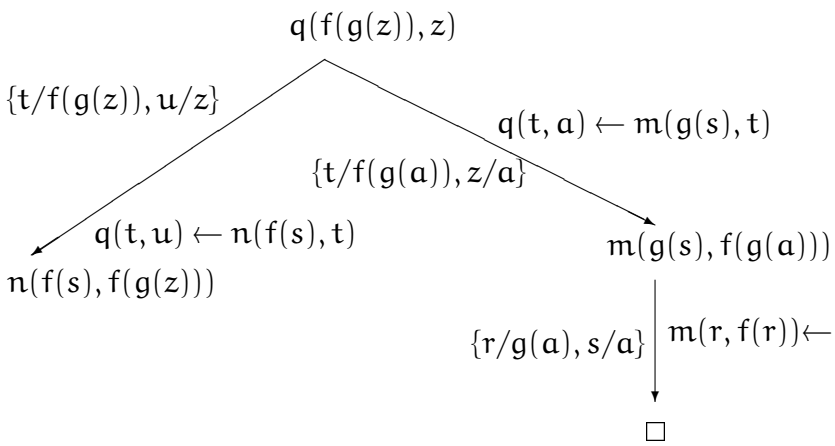

Figure 2.7: The *SLD*-tree corresponding to collection $q(f(g(z)), z) \cdot D_2$ in Example 2.2.5

$$d_4 := q(f(x), z) \xrightarrow[q(t,a) \leftarrow m(g(s),t)]{\{t/f(x), z/a\}} m(g(s), f(x)) \xrightarrow[m(r,f(r)) \leftarrow]{\{r/g(s), x/g(s)\}} \square$$

$$d_5 := n(x) \xrightarrow[n(t) \leftarrow p(t,t)]{\{t/x\}} p(x, x)$$

$$d_6 := q(f(x), z) \xrightarrow[q(t,a) \leftarrow m(g(s),t)]{\{t/f(x), z/a\}} m(g(s), f(x))$$

$$d_7 := q(x, y) \xrightarrow[q(t,u) \leftarrow n(f(s),t)]{\{t/x, u/y\}} n(f(s), x)$$

A representation of D_1 and D_2 is in Figures 2.2 and 2.3. The following hold.

$$q(f(g(z)), z) \cdot D_2 = q(f(g(z)), z) \mapsto \mathit{prefix}(\partial_{\{x/g(z)\}}(d_4)) \cup \left\{\partial_{\{x/f(g(z)), y/z\}}(d_2)\right\}$$

$$D_1 \triangleright D_2 = \begin{cases} p(x, y) \mapsto \mathit{prefix}(d_1 :: d_2) \cup \{d_3\} \\ n(x) \quad \mapsto \{n(x);\ d_5\} \end{cases}$$

$$D_1 \times D_2 = \begin{cases} p(x, y), q(x, y) & \mapsto \mathit{prefix}(d_3 \wedge d_2) \cup \{d_1 \wedge d_2\} \\ p(x, y), q(f(x), y) & \mapsto \mathit{prefix}(d_3 \wedge d_4) \cup \{d_1 \wedge d_4\} \\ n(x), q(x, y) & \mapsto \{n(x), q(x, y);\ d_5 \wedge d_2\} \\ n(x), q(f(x), y) & \mapsto \{n(x), q(f(x), y);\ d_5 \wedge d_4\} \end{cases}$$

where (as in Example 2.2.2 on page 45)

$$\partial_{\{x/f(g(z)), y/z\}}(d_2) = q(f(g(z)), z) \xrightarrow[q(t,u) \leftarrow n(f(s),t)]{\{t/f(g(z)), u/z\}} n(f(s), f(g(z)))$$

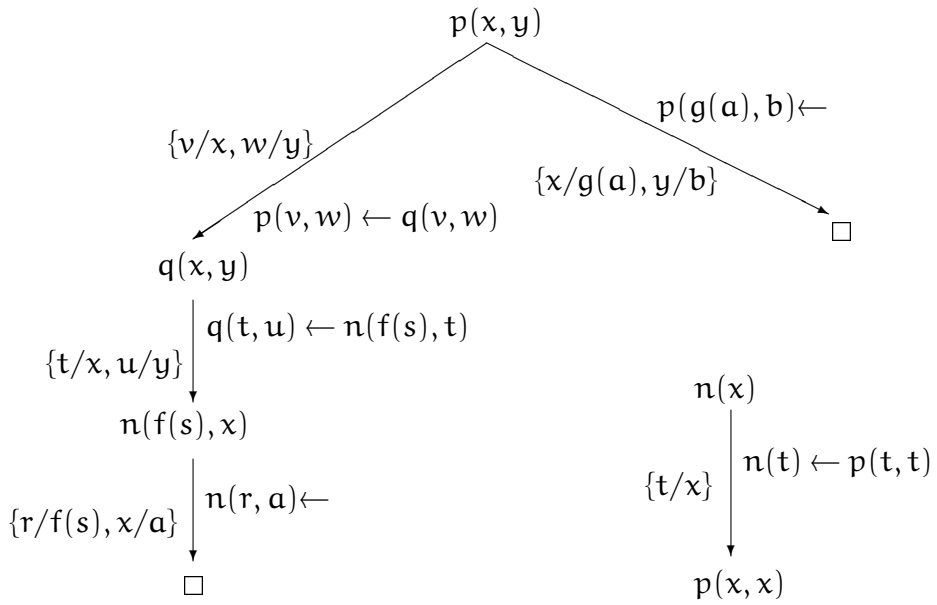

Figure 2.8: The *SLD*-trees corresponding to collection $D_1 \triangleright D_2$ in Example 2.2.5

$$\partial_{\{x/g(z)\}}(d_4) = q(f(g(z)), z) \xrightarrow[q(t,a)\leftarrow m(g(s),t)]{\{t/f(g(a)),z/a\}} m(g(s), f(g(a))) \xrightarrow[m(r,f(r))\leftarrow]{\{r/g(a),s/a\}} \square$$

and

$$d_1 :: d_2 = p(x, y) \xrightarrow[p(v,w)\leftarrow q(v,w)]{\{v/x,w/y\}} q(x, y) \xrightarrow[q(t,u)\leftarrow n(f(s),t)]{\{t/x,u/y\}} n(f(s), x) \xrightarrow[n(r,a)\leftarrow]{\{r/f(s),x/a\}} \square$$

$$d_1 \wedge d_2 = p(x, y), q(x, y) \xrightarrow[p(v,w)\leftarrow q(v,w)]{\{v/x,w/y\}} q(x, y), q(x, y)$$

$$d_3 \wedge d_2 = p(x, y), q(x, y) \xrightarrow[p(g(a),b)\leftarrow]{\{x/g(a),y/b\}} q(g(a), b) \xrightarrow[q(t,u)\leftarrow n(f(s),t)]{\{t/g(a),u/b\}} n(f(s), g(a))$$

$$d_3 \wedge d_4 = p(x, y), q(f(x), z) \xrightarrow[p(g(a),b)\leftarrow]{\{x/g(a),y/b\}} q(f(g(a)), z) \xrightarrow[q(t,a)\leftarrow m(g(s),t)]{\{t/f(g(a)),z/a\}} m(g(s), f(g(a))) \xrightarrow[m(r,f(r))\leftarrow]{\{r/g(a),s/a\}} \square$$

$$d_1 \wedge d_4 = p(x, y), q(f(x), z) \xrightarrow[p(v,w)\leftarrow q(v,w)]{\{v/x,w/y\}} q(x, y), q(f(x), z)$$

$$d_5 \wedge d_2 = n(x), q(x, y) \xrightarrow[n(t)\leftarrow p(t,t)]{\{t/x\}} p(x, x), q(x, y)$$

$$d_5 \wedge d_4 = n(x), q(f(x), z) \xrightarrow[n(t)\leftarrow p(t,t)]{\{t/x\}} p(x, x), q(f(x), z)$$

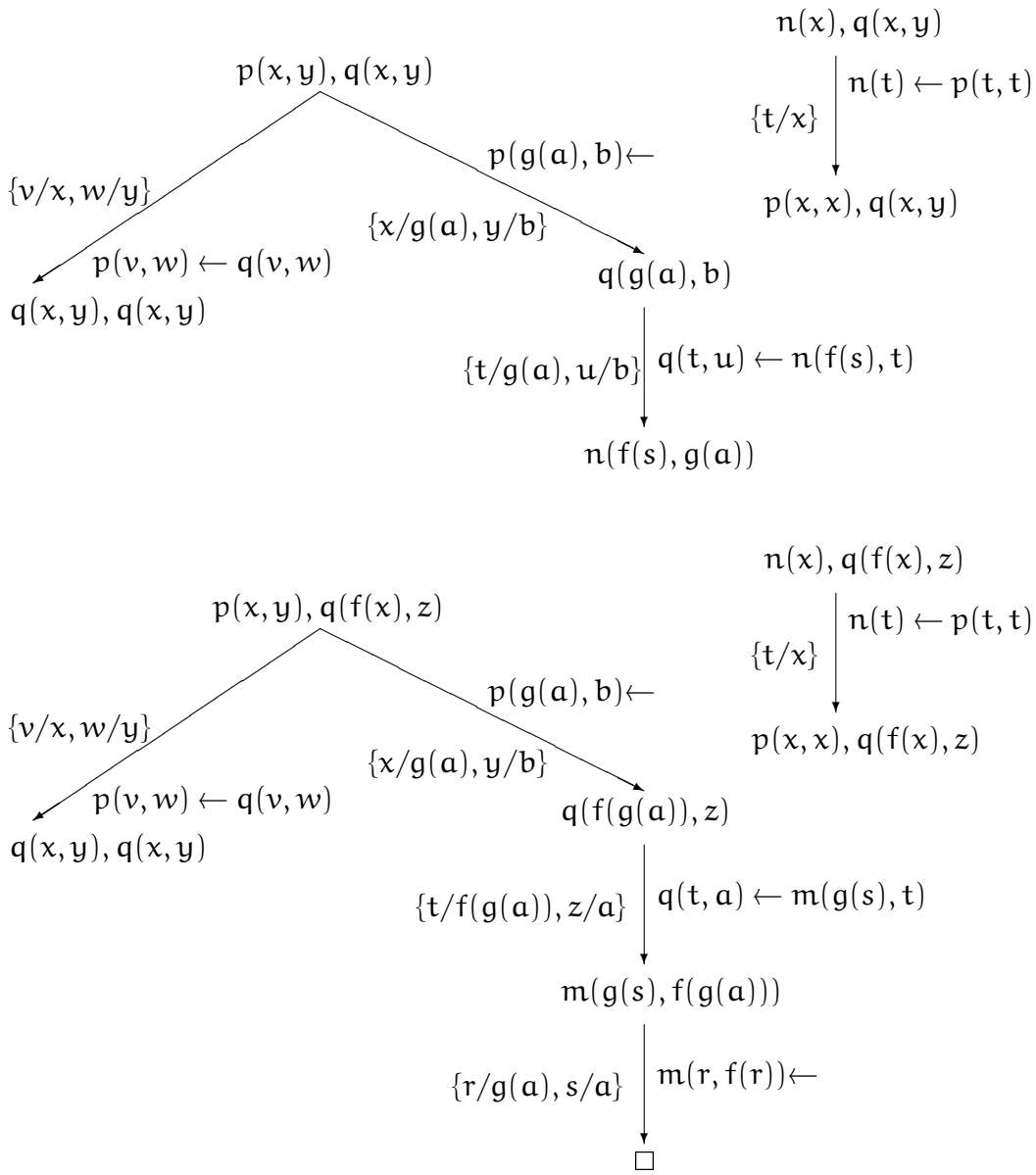

Figure 2.9: The *SLD*-trees corresponding to collection $D_1 \times D_2$ in Example 2.2.5

See Figures 2.7, 2.8 and 2.9 for the corresponding *SLD*-tree representations.

Note that $\{\mathtt{p}(\mathtt{x},\mathtt{y});\ \mathtt{d}_1;\ \mathtt{d}_3\} \subset \mathit{prefix}(\mathtt{d}_1 :: \mathtt{d}_2) \cup \{\mathtt{d}_3\}$ and that, even if $\{\mathtt{d}_1 \wedge \mathtt{d}_4\}$ it *is not* a well-formed set of derivations, the set $\mathit{prefix}(\mathtt{d}_3 \wedge \mathtt{d}_4) \cup \{\mathtt{d}_1 \wedge \mathtt{d}_4\}$ it *is*.

It is important to note that $\equiv_{\mathbb{C}}$ is a congruence w.r.t. $\cdot$, $\times$, $\triangleright$ and $\sum$ because of the renaming apart property and the "collecting" nature of these operations. Furthermore, for each $A \in \mathit{Atoms}$, $D, D' \in \mathbb{C}$ such that $D \equiv_{\mathbb{C}} D'$,

$$A \cdot D = A \cdot D'. \tag{2.2.6}$$

Now we prove some properties of the operations on collections that will be useful in the sequel.

Lemma 2.2.6 *Let* $D_1, D_2, D_3 \in \mathbb{C}$. *Then*

1. $D_1 \times (D_2 \times D_3) = (D_1 \times D_2) \times D_3$
2. *If* $D_3 \sqsubseteq D_2$ *then* $D_1 \triangleright (D_2 \triangleright D_3) = (D_1 \triangleright D_2) \triangleright D_3$.

Proof. The proof of Point 1 follows by Point 2 of Lemma 2.2.4 and by definition of $\times$. To prove Point 2, first of all observe that, by definition of $\triangleright$, for any $D_1, D_2, D_3 \in \mathbb{C}$, $(D_1 \triangleright D_2) \triangleright D_3 = D_1 \triangleright ((D_2 \triangleright D_3) + D_3)$. Moreover, if $D_3 \sqsubseteq D_2$, $D_3 \sqsubseteq D_2 \triangleright D_3$ and then, since $+$ is the *lub* operation on $\mathbb{C}$, $(D_2 \triangleright D_3) + D_3 = D_2 \triangleright D_3$. Therefore, $D_1 \triangleright ((D_2 \triangleright D_3) + D_3) = D_1 \triangleright (D_2 \triangleright D_3)$. ■

Note that, by Lemma 2.2.6 and by a straightforward inductive argument, for any $n \geq 2$ and $D \in \mathbb{C}$,

$$\underbrace{D \triangleright (D \triangleright (\cdots \triangleright D))}_{n} = \underbrace{((D \triangleright D) \triangleright \cdots) \triangleright D}_{n}. \tag{2.2.7}$$

Therefore, we can omit the parentheses in such formulas.

Lemma 2.2.7 $\cdot$, $\times$ *and* $\triangleright$ *distribute over sums in* $(\mathbb{C}, \sqsubseteq)$.

Proof. We have to prove that, for any $\{D_j\}_{j \in J} \subseteq \mathbb{C}$ and $D \in \mathbb{C}$, $\sum\{D_j \triangleright D\}_{j \in J} = (\sum\{D_j\}_{j \in J}) \triangleright D$ and $D \triangleright (\sum\{D_j\}_{j \in J}) = \sum\{D \triangleright D_j\}_{j \in J}$. Analogously for $\cdot$ and $\times$. The proof is straightforward by observing that $\triangleright$, $\cdot$ and $\times$ are defined by collecting the results of operations defined on single derivations. ■

Since the *lub* operation on $(\mathbb{C}, \sqsubseteq)$ coincides with $\sum$, a straightforward consequence is that $\cdot$, $\times$ and $\triangleright$ are continuous (and monotonic) on $(\mathbb{C}, \sqsubseteq)$.

2.3 Operational Semantics

The operational semantics of queries in the program P can be described in terms of the following transition system $\mathcal{T} := (\mathbb{C}, \underset{\mathsf{P}}{\longmapsto})$. Since we want the rules of $\mathcal{T}$ do depend on properties of well-formed sets, rather than on the structure of a *single* derivation step, we can use the rule

$$\frac{\mathsf{D} \in \mathbb{C}, \quad \mathsf{D} \neq \mathsf{D} \triangleright su(tree(\mathsf{P}))}{\mathsf{D} \underset{\mathsf{P}}{\longmapsto} \mathsf{D} \triangleright su(tree(\mathsf{P}))} \tag{2.3.1}$$

where, for any pure collection D,

$$su(\mathsf{D}) := \sum \left\{(\mathsf{A} \cdot \mathsf{D}) \times Id_{\mathbb{C}}\right\}_{\mathsf{A} \in Atoms}. \tag{2.3.2}$$

Note that $su(\mathsf{D})$ can be viewed as the *sequential unfolding* of the pure collection D and it is closed under renaming and under instantiation, since we consider all the possible evaluations of D. Note that we use the construction $\cdot \times Id_{\mathbb{C}}$ to allow the construction $\cdot \triangleright su(tree(\mathsf{P}))$ to extend all derivations in the range of D whose last goal leftmost atom matches the head of a clause in P.

The *initial states* of $\mathcal{T}$ are all the collections of *SLD*-derivations of length zero, while the *final states* are the collections of *SLD*-refutations and finite failures.

As the intuition suggests, the transition system $\mathcal{T}$ defines the usual notion of *SLD*-derivation. The formal statement is given in Theorem 2.3.2. The specificity of this transition system is due to the fact that *we have defined it using the same semantic operators used in the denotational definition.*

Example 2.3.1

Consider the (slight modification[6] of the well-known *ancestor*) program P of Figure 2.10 and the collection $\mathsf{D} := tree(\mathsf{P})$ which is (see Figure 2.11)

$$\begin{cases} \mathsf{anc}(x,y) \mapsto \{\mathsf{anc}(x,y);\ \mathsf{d_a};\ \mathsf{d_b}\} \\ \mathsf{par}(x,y) \mapsto \{\mathsf{par}(x,y);\ \mathsf{d_c};\ \mathsf{d_e}\} \end{cases}$$

where

$$\mathsf{d_a} := \mathsf{anc}(x,y) \xrightarrow[\mathsf{anc}(v,w) \leftarrow \mathsf{par}(v,w)]{\{v/x, w/y\}} \mathsf{par}(x,y)$$

$$\mathsf{d_b} := \mathsf{anc}(x,y) \xrightarrow[\mathsf{anc}(v,w) \leftarrow \mathsf{par}(v,z), \mathsf{anc}(z,w)]{\{v/x, w/y\}} \mathsf{par}(x,z), \mathsf{anc}(z,y)$$

$$\mathsf{d_c} := \mathsf{par}(x,y) \xrightarrow[\mathsf{par}(\mathsf{joe},\mathsf{jay}) \leftarrow]{\{x/\mathsf{joe}, y/\mathsf{jay}\}} \square$$

[6] I shortened all predicates names to limit the size of the formulas.

```
c1: anc(X,Y) :- par(X,Y).
c2: anc(X,Y) :- par(X,Z), anc(Z,Y).
c3: par(joe,jay).
c4: par(jay,tim).
```

Figure 2.10: The ancestor program

$$\mathtt{d_e} := \mathtt{par}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{par(jay,tim)}\leftarrow]{\{\mathtt{x/jay},\mathtt{y/tim}\}} \square$$

Furthermore, consider the collection $\mathsf{D}' := \phi_{\mathtt{anc(x,y)}}$ and let $\mathsf{D}'' := \mathsf{D}\left[{}^{\aleph}/_{\mathtt{par(x,y)}}\right]$. Since, for each pure atom $\mathsf{p}(\mathbf{x})$, $\mathsf{D}(\mathsf{p}(\mathbf{x})) = \mathit{su}(\mathsf{D})(\mathsf{p}(\mathbf{x}))$, it is easy to see that $\mathsf{D}' \underset{\mathsf{P}}{\longmapsto} \mathsf{D}''$.

Now we want to compute the collection $\mathsf{D}'' \triangleright \mathit{su}(\mathsf{D})$. First of all note that, if for a collection $\bar{\mathsf{D}}$ the set $\bar{\mathsf{D}}(\mathbf{G})$ is finite, for any goal $\mathbf{G}$, the operation $\bar{\mathsf{D}} \triangleright \mathit{su}(\mathit{tree}(\mathsf{P}))$ can use only a *finite* part of $\mathit{su}(\mathit{tree}(\mathsf{P}))$ to extend $\bar{\mathsf{D}}$. Hence it is quite easy to build a finite collection $\mathsf{D}''' \sqsubset \mathit{su}(\mathsf{D})$ such that $\mathsf{D}'' \triangleright \mathit{su}(\mathsf{D}) = \mathsf{D}'' \triangleright \mathsf{D}'''$. It is the collection

$$\mathsf{D}''' := \begin{cases} \mathtt{par}(\mathtt{x},\mathtt{z}), \mathtt{anc}(\mathtt{z},\mathtt{y}) \mapsto \mathit{prefix}(\mathtt{d_a}) \cup \left\{\mathtt{d_b}\right\} \\ \mathtt{par}(\mathtt{x},\mathtt{y}) \qquad\qquad\qquad \mapsto \left\{\mathtt{par}(\mathtt{x},\mathtt{y});\ \ \mathtt{d_c};\ \ \mathtt{d_e}\right\} \end{cases}$$

where

$$\mathtt{d_a} := \mathtt{par}(\mathtt{x},\mathtt{z}), \mathtt{anc}(\mathtt{z},\mathtt{y}) \xrightarrow[\mathtt{par(joe,jay)}\leftarrow]{\{\mathtt{x/joe},\mathtt{z/jay}\}} \mathtt{anc}(\mathtt{jay},\mathtt{y})$$

$$\mathtt{d_b} := \mathtt{par}(\mathtt{x},\mathtt{z}), \mathtt{anc}(\mathtt{z},\mathtt{y}) \xrightarrow[\mathtt{par(jay,tim)}\leftarrow]{\{\mathtt{x/jay},\mathtt{z/tim}\}} \mathtt{anc}(\mathtt{tim},\mathtt{y})$$

$$\mathtt{d_c} := \mathtt{par}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{par(joe,jay)}\leftarrow]{\{\mathtt{x/joe},\mathtt{y/jay}\}} \square$$

$$\mathtt{d_e} := \mathtt{par}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{par(jay,tim)}\leftarrow]{\{\mathtt{x/jay},\mathtt{y/tim}\}} \square$$

Since $\mathsf{D}'' \triangleright \mathsf{D}''' \sqsupset \mathsf{D}''$, we can do the transition $\mathsf{D}'' \underset{\mathsf{P}}{\longmapsto} \mathsf{D}''''$, where

$$\mathsf{D}'''' := \mathsf{D}'' \triangleright \mathsf{D}''' = \mathtt{anc}(\mathtt{x},\mathtt{y}) \mapsto \mathit{prefix}(\mathtt{d_f}) \cup \left\{\mathtt{d_g};\ \ \mathtt{d_h};\ \ \mathtt{d_i}\right\}$$

and

$$\mathtt{d_f} := \mathtt{anc}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{anc}(v,w)\leftarrow\mathtt{par}(v,w)]{\{v/\mathtt{x},w/\mathtt{y}\}} \mathtt{par}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{par(joe,jay)}\leftarrow]{\{\mathtt{x/joe},\mathtt{y/jay}\}} \square$$

$$\mathtt{d_g} := \mathtt{anc}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{anc}(v,w)\leftarrow\mathtt{par}(v,w)]{\{v/\mathtt{x},w/\mathtt{y}\}} \mathtt{par}(\mathtt{x},\mathtt{y}) \xrightarrow[\mathtt{par(jay,tim)}\leftarrow]{\{\mathtt{x/jay},\mathtt{y/tim}\}} \square$$

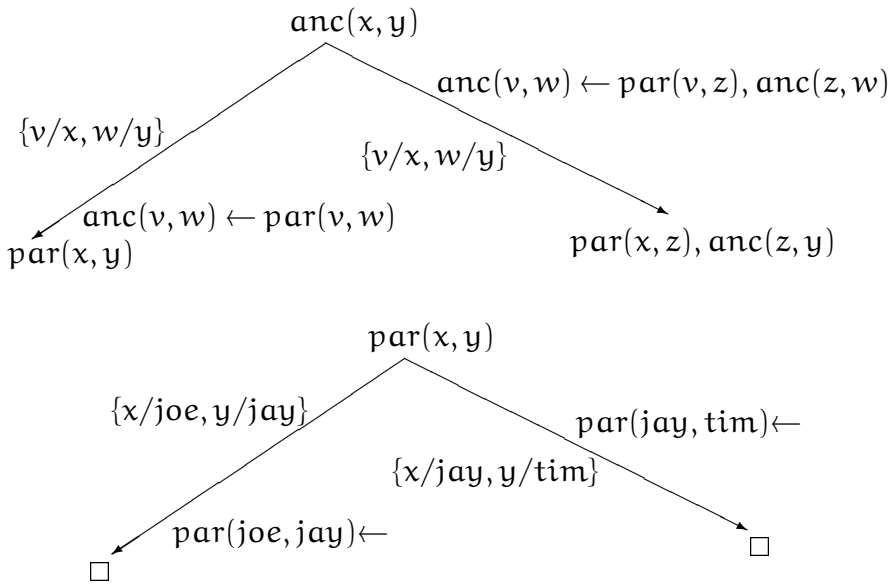

Figure 2.11: The *SLD*-trees corresponding to collection D of Example 2.3.1

$$\begin{aligned}
d_h := \mathrm{anc}(x,y) &\xrightarrow[\mathrm{anc}(v,w)\leftarrow \mathrm{par}(v,z),\mathrm{anc}(z,w)]{\{v/x,w/y\}} \mathrm{par}(x,z),\mathrm{anc}(z,y) \\
&\xrightarrow[\mathrm{par}(joe,jay)\leftarrow]{\{x/joe,z/jay\}} \mathrm{anc}(jay,y) \\
d_i := \mathrm{anc}(x,y) &\xrightarrow[\mathrm{anc}(v,w)\leftarrow \mathrm{par}(v,z),\mathrm{anc}(z,w)]{\{v/x,w/y\}} \mathrm{par}(x,z),\mathrm{anc}(z,y) \\
&\xrightarrow[\mathrm{par}(jay,tim)\leftarrow]{\{x/jay,z/tim\}} \mathrm{anc}(tim,y)
\end{aligned}$$

See Figure 2.12 for an *SLD*-tree representation of this transition.

Since we are interested in all the *SLD*-derivations of a query **G** *in* P, we define its *behavior* as

$$\mathcal{B}[\![\mathbf{G}\ \mathit{in}\ P]\!] := \sum \left\{ D \mid \phi_G \underset{P}{\longmapsto}^* D \right\} \tag{2.3.3}$$

where $\underset{P}{\longmapsto}^*$ is the reflexive and transitive closure of $\underset{P}{\longmapsto}$. The behavior (modulo variance) of a query is the *operational semantics* of the query. As we will see in the following (Corollary 2.5.17) every query has equivalent operational and denotational semantics.

The behavior of a query indeed models all *SLD*-trees of that query, as shown by the following theorem.

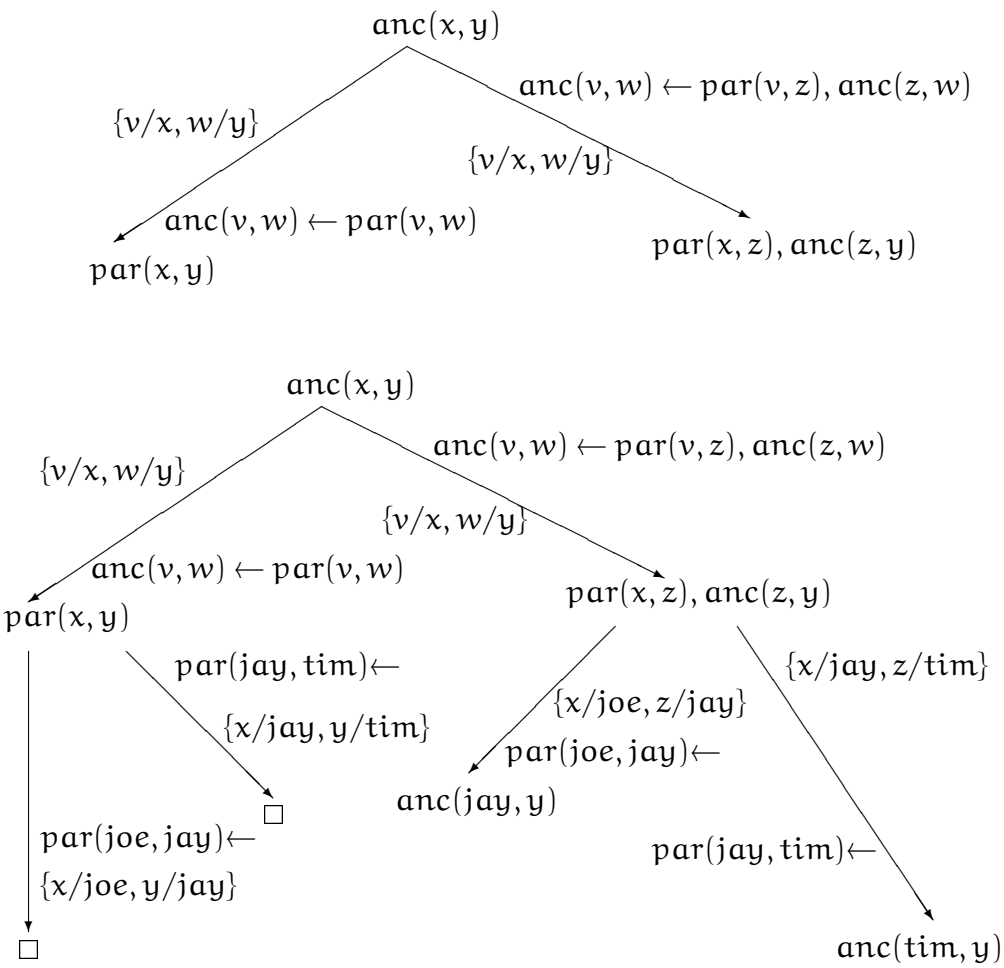

Figure 2.12: We show the *SLD*-tree transition $D'' \underset{P}{\longmapsto} D''''$ of Example 2.3.1, where D'' and D'''' are represented by the upper and lower trees of the figure

Theorem 2.3.2 *Let* P *be a program and* $\mathbf{G}$ *be a goal. Then*

$$\mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!] = \phi\left[\left\{\mathsf{d} \mid \mathsf{d} := \mathbf{G} \xrightarrow[\mathsf{P}]{\vartheta}{}^{*} \mathbf{B}, \mathbf{B} \in Goals\right\} \Big/ \mathbf{G}\right].$$

Proof. To prove the thesis it is sufficient to show that, for any collection D and any goal $\mathbf{G}$, such that $\mathsf{D} \underset{\mathsf{P}}{\longmapsto} \mathsf{D} \triangleright su(tree(\mathsf{P}))$, $\bar{\mathsf{d}} \in (\mathsf{D} \triangleright su(tree(\mathsf{P})))(\mathbf{G}) \setminus \mathsf{D}(\mathbf{G})$ if and only if $\bar{\mathsf{d}} = \mathbf{G} \xrightarrow[\mathsf{c}_1]{\vartheta_1} \cdots \xrightarrow[\mathsf{c}_n]{\vartheta_n} (\mathsf{A}, \mathbf{G}') \xrightarrow[\mathsf{c}']{\sigma} (\mathbf{B}', \mathbf{G}')\sigma$, where $\mathbf{G} \xrightarrow[\mathsf{c}_1]{\vartheta_1} \cdots \xrightarrow[\mathsf{c}_n]{\vartheta_n} (\mathsf{A}, \mathbf{G}') =: \mathsf{d} \in \mathsf{D}(\mathbf{G})$, $\mathsf{c}' = \mathsf{p}(\mathbf{t}') \leftarrow \mathbf{B}'$ is a renamed apart (w.r.t. d) version of a clause $\mathsf{c} \in \mathsf{P}$ and $\sigma = mgu(\mathsf{A}, \mathsf{p}(\mathbf{t}'))$. The proof follows then by definition of $\mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!]$, by definition of derivation and by a straightforward inductive argument. We prove the two implications separately.

$\Longrightarrow$ By definition of su, $\triangleright$ and since $su(\mathsf{D})$ is closed under renaming, there exists a derivation $\mathsf{d} = \mathbf{G} \xrightarrow[\mathsf{c}_1]{\vartheta_1} \cdots \xrightarrow[\mathsf{c}_n]{\vartheta_n} (\mathsf{A}, \mathbf{G}') \in \mathsf{D}$ and there exists a non-null derivation $\mathsf{d}_1 \in ((\mathsf{A} \cdot tree(\mathsf{P})) \times \phi_{\mathbf{G}'})(\mathsf{A}, \mathbf{G}')$ such that

$$\bar{\mathsf{d}} = \mathsf{d} :: \mathsf{d}_1 \text{ and } var(\mathsf{d}) \cap var(\mathsf{d}_1) = var(\mathsf{A}, \mathbf{G}'). \tag{1}$$

By definition of $\times$, there exists

$$\mathsf{d}_2 \in (\mathsf{A} \cdot tree(\mathsf{P}))(\mathsf{A}) \setminus \{\mathsf{A}\} \text{ such that } \mathsf{d}_1 = \mathsf{d}_2 \wedge \mathbf{G}'. \tag{2}$$

Moreover, by definition of $\cdot$ and $tree$, $\mathsf{d}_2 = \partial_\delta(\mathsf{d}_c)$, where $\mathsf{d}_c = \mathsf{p}(\mathbf{x}) \xrightarrow[\mathsf{c}\rho]{\{\mathbf{x}/\mathbf{t}\rho\}} \mathbf{B}\rho$, $\mathsf{c} = \mathsf{p}(\mathbf{t}) \leftarrow \mathbf{B} \in \mathsf{P}$, $\mathbf{x}$ is a tuple of new distinct variables, $var(\mathsf{A}) \cap var(\mathsf{c}\rho) = \emptyset$ and δ is an idempotent substitution such that $\mathsf{A} = \mathsf{p}(\mathbf{x})\delta$. Then, by definition of ∂ and since $\mathsf{d}_2 \neq \mathsf{A}$, $\mathsf{d}_2 = \mathsf{A} \xrightarrow[\mathsf{c}\rho]{\sigma} \mathbf{B}\rho\sigma$, where $\sigma = mgu(\mathsf{A}, \mathsf{p}(\mathbf{t})\rho)$. Moreover, by (1) and (2), $var(\mathsf{d}) \cap var(\mathsf{c}\rho) = \emptyset$. Then, by (2) and by definition of $\wedge$, $\mathsf{d}_1 = (\mathsf{A}, \mathbf{G}') \xrightarrow[\mathsf{c}\rho]{\sigma} \mathbf{B}\rho\sigma, \mathbf{G}'\sigma$. Hence, $\bar{\mathsf{d}} = \mathbf{G} \xrightarrow[\mathsf{c}_1]{\vartheta_1} \cdots \xrightarrow[\mathsf{c}_n]{\vartheta_n} (\mathsf{A}, \mathbf{G}') \xrightarrow[\mathsf{c}\rho]{\sigma} (\mathbf{B}\rho, \mathbf{G}')\sigma$. Now, to complete it is sufficient to observe that $\mathsf{c}\rho$ is a renamed apart (w.r.t. d) version of $\mathsf{c} \in \mathsf{P}$.

$\Longleftarrow$ By definition of derivation, $var(\mathsf{c}') \cap var(\mathsf{A}) = \emptyset$. Then, by definition of ∂, there exists $\mathsf{d}_1 = \partial_\delta(\mathsf{p}(\mathbf{x}) \xrightarrow[\mathsf{c}']{\{\mathbf{x}/\mathbf{t}'\}} \mathbf{B}')$, where $\mathbf{x}$ is a tuple of new distinct variables and δ is an idempotent substitution such that $\mathsf{A} = \mathsf{p}(\mathbf{x})\delta$. Moreover, by definition of $tree$ and $\cdot$, $\mathsf{d}_1 \in (\mathsf{A} \cdot tree(\mathsf{c}))(\mathsf{A})$. Therefore, since $\mathsf{c} \in \mathsf{P}$, by definition of $tree$ and since $\cdot$ is monotonic, $\mathsf{d}_1 \in (\mathsf{A} \cdot tree(\mathsf{P}))(\mathsf{A})$. Then, by definition of $\times$, $\mathsf{d}' = \mathsf{d}_1 \wedge \mathbf{G}' = (\mathsf{A}, \mathbf{G}') \xrightarrow[\mathsf{c}']{\sigma} (\mathbf{B}'\sigma, \mathbf{G}'\sigma) \in ((\mathsf{A} \cdot tree(\mathsf{P})) \times Id_{\mathbb{C}})(\mathsf{A}, \mathbf{G}') = su(tree(\mathsf{P}))(\mathsf{A}, \mathbf{G}')$. Moreover, by definition of derivation, $var(\mathsf{d}) \cap var(\mathsf{d}') = var(\mathsf{A}, \mathbf{G}')$. Then to prove the thesis it is sufficient

to observe that, by definition of $\triangleright$, since $\mathtt{d} \in \mathsf{D}(\mathbf{G})$ and $\mathit{last}(\mathtt{d}) = \mathit{first}(\mathtt{d}')$, $\bar{\mathtt{d}} = \mathtt{d} :: \mathtt{d}' \in (\mathsf{D} \triangleright \mathit{su}(\mathit{tree}(\mathsf{P})))(\mathbf{G})$. ■

2.4 The Program Denotation

From the notion of query behavior, we can define the behavior of a program as the collection $\sum\{\mathcal{B}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}]\!]\}_{\mathbf{G}\in \mathit{Goals}}$. This collection can be viewed as a program denotation but we can define a better top-down program denotation. This can be obtained[7] by collecting *only* the behaviors for all pure atomic goals, i.e., the behaviors of the procedures with no constraints on the inputs. This yields a compact denotation which is a finite-support function (that may give infinite results).

The *top-down SLD-tree denotation* of a program P is the interpretation

$$\mathcal{O}[\![\mathsf{P}]\!] := \sum \left\{ \mathcal{B}[\![\mathsf{p}(\mathbf{x})\ \mathit{in}\ \mathsf{P}]\!]/_{\equiv_{\mathbb{C}}} \right\}_{\mathsf{p}(\mathbf{x})\in \mathit{Goals}}. \tag{2.4.1}$$

This can be viewed as a *program* denotation, since it is the semantics of the program as a set of definite clauses (or a set of procedure definitions).

Using standard techniques it can be proved that $\mathcal{P}[\![\mathsf{P}]\!]$ is continuous; hence, we can define the *fixpoint SLD-tree denotation* of the program P as the interpretation

$$\mathcal{F}[\![\mathsf{P}]\!] := \mathit{lfp}\, \mathcal{P}[\![\mathsf{P}]\!]. \tag{2.4.2}$$

In the following we will prove that $\mathcal{F}[\![\mathsf{P}]\!] = \mathcal{O}[\![\mathsf{P}]\!]$.

Example 2.4.1

We give now an example to present some nice properties of our base semantic, which have the same flavor of those of s-semantics approach.

Consider the toy program P of Figure 2.13. By definition of $\mathcal{P}[\![\cdot]\!]$,

$$\mathcal{P}[\![\mathsf{P}]\!]\uparrow_1 = \mathcal{P}[\![\mathsf{P}]\!]_\phi = \mathsf{q}(\mathsf{x}) \mapsto \left\{\mathsf{q}(\mathsf{x});\ \ \mathsf{q}(\mathsf{x}) \xrightarrow[\mathsf{q}(\mathsf{f}(\mathsf{y}))\leftarrow]{\{\mathsf{x}/\mathsf{f}(\mathsf{y})\}} \Box\right\} =: \mathrm{I}$$

$$\mathcal{P}[\![\mathsf{P}]\!]\uparrow_2 = \mathcal{P}[\![\mathsf{P}]\!]_\mathrm{I} = \begin{cases} \mathsf{p}(\mathsf{x},\mathsf{y}) \mapsto \mathit{prefix}\left(\mathsf{p}(\mathsf{x},\mathsf{y}) \xrightarrow[\mathsf{p}(v,\mathsf{a})\leftarrow \mathsf{q}(v)]{\{v/\mathsf{x},\mathsf{y}/\mathsf{a}\}} \mathsf{q}(\mathsf{x}) \xrightarrow[\mathsf{q}(\mathsf{f}(w))\leftarrow]{\{\mathsf{x}/\mathsf{f}(w)\}} \Box\right) \\ \mathsf{q}(\mathsf{x}) \quad \mapsto \left\{\mathsf{q}(\mathsf{x});\ \ \mathsf{q}(\mathsf{x}) \xrightarrow[\mathsf{q}(\mathsf{f}(\mathsf{y}))\leftarrow]{\{\mathsf{x}/\mathsf{f}(\mathsf{y})\}} \Box\right\} \end{cases}$$

$$\mathcal{P}[\![\mathsf{P}]\!]\uparrow_3 = \mathcal{P}[\![\mathsf{P}]\!]\uparrow_2 = \mathcal{P}[\![\mathsf{P}]\!]\uparrow_\omega$$

It is interesting to note that the denotation of the program P, even if fully detailed, is finite. This is not the case of a (much much more) abstract semantics as the success set semantics (see Section 1.3.3), which denotes P by the (infinite) set

$$\left\{\mathsf{q}(\mathsf{f}(\mathsf{a}));\ \ \mathsf{p}(\mathsf{f}(\mathsf{a}),\mathsf{a});\ \ \mathsf{q}(\mathsf{f}(\mathsf{f}(\mathsf{a})));\ \ \mathsf{p}(\mathsf{f}(\mathsf{f}(\mathsf{a})),\mathsf{a});\ \ \mathsf{q}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))));\ \ \ldots\right\}$$

[7] Essentially because of Corollary 2.5.5.

```
c1: q(f(X)).
c2: p(X,a) :- q(X).
```

Figure 2.13: The toy program of Example 2.4.1

One of the properties which is involved in such a phenomenon is the *independence from the language*, which means that the denotation is not affected by the choice of the language signature. This property also means that the denotation of a program fragment is determined by the symbols it contains without considering the entire underlying alphabet. The advantage of enjoying such a property becomes more tangible in the case where we wish to actually compute program denotations – as in the case of semantic-based program analysis and diagnosis. Note that the same property does not hold for other variable-based semantics, such as those in [16, 48].

Program denotations are strongly related to program equivalences. We define the equivalence $\approx$ of two programs P_1, P_2 as the equivalence of the behaviors of the two programs, i.e.,

$$\mathsf{P}_1 \approx \mathsf{P}_2 \iff \forall \mathbf{G} \in \mathit{Goals}.\ \mathcal{B}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}_1]\!] = \mathcal{B}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}_2]\!].$$

Now we give two definitions to relate program equivalences to denotations. Let $\mathcal{S}[\![\mathsf{P}]\!]$ be a program denotation and $\sim$ be a program equivalence. Then

1. $\mathcal{S}[\![\cdot]\!]$ is *correct* w.r.t. $\sim$ if $\mathcal{S}[\![\mathsf{P}_1]\!] = \mathcal{S}[\![\mathsf{P}_2]\!] \Longrightarrow \mathsf{P}_1 \sim \mathsf{P}_2$,
2. $\mathcal{S}[\![\cdot]\!]$ is *minimal* w.r.t. $\sim$ if $\mathsf{P}_1 \sim \mathsf{P}_2 \Longrightarrow \mathcal{S}[\![\mathsf{P}_1]\!] = \mathcal{S}[\![\mathsf{P}_2]\!]$.

In the following (Corollary 2.5.6) we will prove that $\mathcal{O}[\![\mathsf{P}]\!]$ (and $\mathcal{F}[\![\mathsf{P}]\!]$) is correct and minimal w.r.t. $\approx$. Note that if a semantics is correct and minimal then it is also the most abstract semantics among the correct ones.

2.5 Semantic Properties of *SLD*-derivations

We show that the program denotation $\mathcal{O}[\![\mathsf{P}]\!]$ has several interesting properties, which can all be viewed as compositionality properties. The first compositionality result is Theorem 2.5.4 which shows that the semantic function $\mathcal{B}[\![\cdot]\!]$ is compositional w.r.t. procedure calls (atomic goals) and composition (conjunction) inside goals (*AND*-compositionality).

Lemma 2.5.1 *Let* A *be an atom,* $\mathsf{D} \in \mathbb{PC}$, $\mathsf{D}', \mathsf{D}'' \in \mathbb{C}$ *and* $\mathbf{G}$ *be a goal. Then*

1. $A \cdot (D' \triangleright su(D)) = (A \cdot D') \triangleright su(D)$.
2. $(D' \triangleright su(D)) \times \phi_G \sqsubseteq (D' \times \phi_G) \triangleright su(D)$.
3. *If* $D' \triangleright su(D) = D'$ *then* $(D' \times D'') \triangleright su(D) = D' \times (D'' \triangleright su(D))$.

Given a pure collection D, we denote by $su_n(D)$ the collection[8]

$$\underbrace{su(D) \triangleright \cdots \triangleright su(D)}_{n}.$$

Thus, the following corollary holds.

Corollary 2.5.2 *Let* P *be a program and* $\mathbf{G}$ *a goal. Then*

1. $\mathcal{B}[\![\mathbf{G} \textit{ in } P]\!] = \phi_G \triangleright \sum\{su_n(tree(P))\}_{n \geq 0}$.
2. $\mathcal{O}[\![P]\!] = (Id_{\mathbb{I}} \triangleright \sum\{su_n(tree(P))\}_{n \geq 0}) \big/_{\equiv_{\mathbb{C}}}$.

Proof. We prove the points separately.

Point 1 We use the notation $D \underset{P}{\longmapsto}^n D'$ to denote that the collection D results in the collection D' with *at most* n transition steps $\underset{P}{\longmapsto}$. We prove (by induction on n) that $\sum\{D' \mid D \underset{P}{\longmapsto}^n D'\} = D \triangleright su_n(tree(P))$. For $n = 0$, $\sum\{D' \mid D \underset{P}{\longmapsto}^0 D'\} = D = D \triangleright su_0(tree(P))$. For $n > 0$ the following facts hold.

$$\begin{array}{ll}
\sum\{D' \mid D \underset{P}{\longmapsto}^n D'\} = & [\text{by definition of } \underset{P}{\longmapsto}^n] \\
\sum\{D' \mid D \underset{P}{\longmapsto}^{n-1} D'', D'' \underset{P}{\longmapsto}^1 D'\} = & [\text{by set theory}] \\
\sum\{\sum\{D' \mid D'' \underset{P}{\longmapsto}^1 D'\} \mid D \underset{P}{\longmapsto}^{n-1} D''\} = & [\text{by definition of } \underset{P}{\longmapsto} \text{ and } tree] \\
\sum\{D'' \triangleright su(tree(P)) \mid D \underset{P}{\longmapsto}^{n-1} D''\} = & [\text{by Lemma 2.2.7}] \\
\sum\{D'' \mid D \underset{P}{\longmapsto}^{n-1} D''\} \triangleright su(tree(P)) = & [\text{by inductive hypothesis}] \\
(D \triangleright su_{n-1}(tree(P))) \triangleright su(tree(P)) = & [\text{by Point 2 of Lemma 2.2.6}] \\
D \triangleright su_n(tree(P)). &
\end{array}$$

Finally

$$\begin{array}{ll}
\mathcal{B}[\![\mathbf{G} \textit{ in } P]\!] = & [\text{by definition}] \\
\sum\{D \mid \phi_G \underset{P}{\longmapsto}^* D\} = & [\text{by definition of } \underset{P}{\longmapsto}^*]
\end{array}$$

[8] Note that $su_1(D) = su(D)$ and we assume that $su_0(D) := \phi$.

$$\sum\{\sum\{\mathsf{D} \mid \phi_{\mathbf{G}} \underset{\mathsf{P}}{\longmapsto}^{n} \mathsf{D}\}\}_{n\geq 0} = \qquad [\text{by the previous result}]$$
$$\sum\{\phi_{\mathbf{G}} \triangleright su_n(tree(\mathsf{P}))\}_{n\geq 0} = \qquad [\text{by Lemma 2.2.7}]$$
$$\phi_{\mathbf{G}} \triangleright \sum\{su_n(tree(\mathsf{P}))\}_{n\geq 0}.$$

Point 2 The following facts hold.

$$\mathcal{O}[\![\mathsf{P}]\!] =$$
$$[\text{by definition of } \mathcal{O}[\![\cdot]\!] \text{ and since } \equiv_{\mathbb{C}} \text{ is a congruence w.r.t. } \textstyle\sum]$$
$$\left(\sum\{\mathcal{B}[\![\mathsf{p}(\mathbf{x}) \ in \ \mathsf{P}]\!]\}_{\mathsf{p}(\mathbf{x})\in Goals}\right) /_{\equiv_{\mathbb{C}}} =$$
$$[\text{by the previous Point 1}]$$
$$\left(\sum\{\phi_{\mathsf{p}(\mathbf{x})} \triangleright \sum\{su_n(tree(\mathsf{P}))\}_{n\geq 0}\}_{\mathsf{p}(\mathbf{x})\in Goals}\right) /_{\equiv_{\mathbb{C}}} =$$
$$[\text{Lemma 2.2.7 and by definition of } Id_{\mathbb{I}}]$$
$$\left(Id_{\mathbb{I}} \triangleright \sum\{su_n(tree(\mathsf{P}))\}_{n\geq 0}\right) /_{\equiv_{\mathbb{C}}}.$$

■

The following technical corollary follows by Lemmata 2.5.1 and 2.2.6 and a straightforward inductive argument.

Corollary 2.5.3 *Let* A *be an atom,* $\mathbf{G}$ *be a goal,* $\mathsf{D} \in \mathbb{PC}$ *and* $\mathsf{D}', \mathsf{D}'' \in \mathbb{C}$. *Then, for any* $n \geq 0$,

1. $\mathsf{A} \cdot (\mathsf{D}' \triangleright su_n(\mathsf{D})) = (\mathsf{A} \cdot \mathsf{D}') \triangleright su_n(\mathsf{D})$.
2. $(\mathsf{D}' \triangleright su_n(\mathsf{D})) \times \phi_{\mathbf{G}} \sqsubseteq (\mathsf{D}' \times \phi_{\mathbf{G}}) \triangleright su_n(\mathsf{D})$.
3. *If* $\mathsf{D}' \triangleright su_n(\mathsf{D}) = \mathsf{D}'$ *then* $(\mathsf{D}' \times \mathsf{D}'') \triangleright su_n(\mathsf{D}) = \mathsf{D}' \times (\mathsf{D}'' \triangleright su_n(\mathsf{D}))$.

Theorem 2.5.4 *Let* A *be an atom,* $\mathbf{G}_1$, $\mathbf{G}_2$ *be goals and* P *be a program. Then*

1. $\mathcal{B}[\![\mathsf{A} \ in \ \mathsf{P}]\!] = \mathsf{A} \cdot \mathcal{O}[\![\mathsf{P}]\!]$. *(the semantics of any atomic goal can be derived from the goal-independent denotation)*
2. $\mathcal{B}[\![(\mathbf{G}_1, \mathbf{G}_2) \ in \ \mathsf{P}]\!] = \mathcal{B}[\![\mathbf{G}_1 \ in \ \mathsf{P}]\!] \times \mathcal{B}[\![\mathbf{G}_2 \ in \ \mathsf{P}]\!]$.

Proof. We prove the points separately.

Point 1 The following equalities hold

$$\mathcal{B}[\![\mathsf{A} \ in \ \mathsf{P}]\!] = \qquad [\text{by Point 1 of Corollary 2.5.2}]$$

$$\begin{aligned}
&\phi_A \triangleright \sum\{su_n(tree(P))\}_{n\geq 0} = && [\text{by definition of } \cdot \text{ and } Id_{\mathbb{I}}]\\
&(A \cdot Id_{\mathbb{I}}) \triangleright \sum\{su_n(tree(P))\}_{n\geq 0} = && [\text{by Corollary 2.5.3 and Lemma 2.2.7}]\\
&A \cdot (Id_{\mathbb{I}} \triangleright \sum\{su_n(tree(P))\}_{n\geq 0}).
\end{aligned}$$

Finally, since (by (2.2.6)) $\equiv_{\mathbb{C}}$ is a congruence w.r.t. $\cdot$ and by Corollary 2.5.2, $A \cdot (Id_{\mathbb{I}} \triangleright \sum\{su_n(tree(P))\}_{n\geq 0}) = A \cdot \mathcal{O}[\![P]\!]$.

Point 2 First of all note that, for any goal $\mathbf{G}$ and any $n \geq 0$,

$$\begin{aligned}
&\mathcal{B}[\![\mathbf{G}\ in\ P]\!] \triangleright su_n(tree(P)) = && [\text{by Corollary 2.5.2}]\\
&(\phi_{\mathbf{G}} \triangleright \sum\{su_k(tree(P))\}_{k\geq 0}) \triangleright su_n(tree(P)) = && [\text{by Lemmata 2.2.6 and 2.2.7}]\\
&\phi_{\mathbf{G}} \triangleright \sum\{su_k(tree(P)) \triangleright su_n(tree(P))\}_{k\geq 0} = && [\text{by (2.2.7)}]\\
&\phi_{\mathbf{G}} \triangleright \sum\{su_k(tree(P))\}_{k\geq 0} = && [\text{by Corollary 2.5.2}]\\
&\mathcal{B}[\![\mathbf{G}\ in\ P]\!].
\end{aligned}$$

Now we prove the two inclusions of the thesis separately.

$\sqsubseteq$ The following facts hold.

$$\begin{aligned}
&\mathcal{B}[\![(\mathbf{G}_1, \mathbf{G}_2)\ in\ P]\!] =\\
&\qquad [\text{by Point 1 of Corollary 2.5.2 and by definition of } \times]\\
&(\phi_{\mathbf{G}_1} \times \phi_{\mathbf{G}_2}) \triangleright \sum\{su_n(tree(P))\}_{n\geq 0} \sqsubseteq\\
&\qquad [\text{since } \times \text{ and } \triangleright \text{ are monotonic and } \phi_{\mathbf{G}_1} \sqsubseteq \mathcal{B}[\![\mathbf{G}_1\ in\ P]\!]]\\
&(\mathcal{B}[\![\mathbf{G}_1\ in\ P]\!] \times \phi_{\mathbf{G}_2}) \triangleright \sum\{su_n(tree(P))\}_{n\geq 0} =\\
&\qquad [\text{by Point 3 of Corollary 2.5.3 and by Lemma 2.2.7}]\\
&\mathcal{B}[\![\mathbf{G}_1\ in\ P]\!] \times (\phi_{\mathbf{G}_2} \triangleright \sum\{su_n(tree(P))\}_{n\geq 0}) =\\
&\qquad [\text{by Corollary 2.5.2}]\\
&\mathcal{B}[\![\mathbf{G}_1\ in\ P]\!] \times \mathcal{B}[\![\mathbf{G}_2\ in\ P]\!].
\end{aligned}$$

$\sqsupseteq$ The following facts hold.

$$\begin{aligned}
&\mathcal{B}[\![\mathbf{G}_1\ in\ P]\!] \times \mathcal{B}[\![\mathbf{G}_2\ in\ P]\!] =\\
&\qquad [\text{by repeating the previous steps}]\\
&(\mathcal{B}[\![\mathbf{G}_1\ in\ P]\!] \times \phi_{\mathbf{G}_2}) \triangleright \sum\{su_n(tree(P))\}_{n\geq 0} =\\
&\qquad [\text{by Corollary 2.5.2}]
\end{aligned}$$

$$((\phi_{\mathbf{G}_1} \triangleright \sum\{su_k(tree(\mathsf{P}))\}_{k\geq 0}) \times \phi_{\mathbf{G}_2}) \triangleright \sum\{su_n(tree(\mathsf{P}))\}_{n\geq 0} \sqsubseteq$$

[by Point 2 of Corollary 2.5.3 and by Lemma 2.2.7]

$$((\phi_{\mathbf{G}_1} \times \phi_{\mathbf{G}_2}) \triangleright \sum\{su_k(tree(\mathsf{P}))\}_{k\geq 0}) \triangleright \sum\{su_n(tree(\mathsf{P}))\}_{n\geq 0} \sqsubseteq$$

[by definition of $\times$ and by Point 1 of Corollary 2.5.2]

$$\mathcal{B}[\![(\mathbf{G}_1, \mathbf{G}_2)\ in\ \mathsf{P}]\!] \triangleright \sum\{su_n(tree(\mathsf{P}))\}_{n\geq 0} =$$

[by Lemma 2.2.7 and by the previous observation]

$$\mathcal{B}[\![(\mathbf{G}_1, \mathbf{G}_2)\ in\ \mathsf{P}]\!].$$

■

The property expressed in Point 1 is sometimes referred to as *condensing* in the program analysis field. It essentially shows that the behavior of any atomic goal can be derived from the goal-independent denotation $\mathcal{O}[\![\mathsf{P}]\!]$.

Essentially because of Theorem 2.5.4 we can always reconstruct an *SLD*-tree for a generic (non-pure and non-atomic) goal from the *SLD*-trees of pure atoms. Indeed, we can immediately derive the following result.

Corollary 2.5.5 (*AND*-compositionality) *Let* A *be an atom,* $\mathbf{G}$ *be a goal and* P *be a program. Then*

1. $\mathcal{B}[\![\Box\ in\ \mathsf{P}]\!] = \phi_{\Box}$,
2. $\mathcal{B}[\![(\mathsf{A}, \mathbf{G})\ in\ \mathsf{P}]\!] = (\mathsf{A} \cdot \mathcal{O}[\![\mathsf{P}]\!]) \times \mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!]$.

This property is usually called *AND-compositionality* and shows that the behavior of any (conjunctive) goal can be derived from the goal-independent denotation $\mathcal{O}[\![\mathsf{P}]\!]$, i.e., from the behaviors of (finitely many) pure atomic goals. It is the property which allows us to take $\mathcal{O}[\![\mathsf{P}]\!]$ as *the* semantics of a program, without being concerned with the behaviors for all possible goals. Indeed, by applying repeatedly Corollary 2.5.5,

$$\mathcal{B}[\![\mathsf{A}_1, \ldots, \mathsf{A}_n\ in\ \mathsf{P}]\!] = \mathsf{A}_1 \cdot \mathcal{O}[\![\mathsf{P}]\!] \times \cdots \times \mathsf{A}_n \cdot \mathcal{O}[\![\mathsf{P}]\!]. \tag{2.5.1}$$

This closure property of $\mathcal{O}[\![\cdot]\!]$ w.r.t. $\mathcal{B}[\![\cdot]\!]$ allows us to show that the denotation $\mathcal{O}[\![\cdot]\!]$ is both correct and minimal w.r.t. $\approx$.

Corollary 2.5.6 *Let* P_1, P_2 *be two programs. Then* $\mathsf{P}_1 \approx \mathsf{P}_2 \Longleftrightarrow \mathcal{O}[\![\mathsf{P}_1]\!] = \mathcal{O}[\![\mathsf{P}_2]\!]$.

Proof. The proof of the implication $\Longrightarrow$ is straightforward by definition of $\approx$ and of $\mathcal{O}[\![\cdot]\!]$. The proof of the other implication is by contradiction. Assume that $\mathsf{P}_1 \not\approx \mathsf{P}_2$ and $\mathcal{O}[\![\mathsf{P}_1]\!] = \mathcal{O}[\![\mathsf{P}_2]\!]$. By definition of $\approx$, there exists $\mathbf{G} \in \mathit{Goals}$ such that $\mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}_1]\!] \neq \mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}_2]\!]$. Now the proof is by structural induction on $\mathbf{G}$.

$\underline{\mathbf{G} = \square}$ Contradictory, since by Point 1 of Corollary 2.5.5, $\mathcal{B}[\![\square \ in \ \mathsf{P}_1]\!] = \phi_\square = \mathcal{B}[\![\square \ in \ \mathsf{P}_2]\!]$.

$\underline{\mathbf{G} = (\mathbf{A}, \mathbf{G}')}$ By Point 2 of Corollary 2.5.5 two cases arise. If $\mathsf{A} \cdot \mathcal{O}[\![\mathsf{P}_1]\!] \neq \mathsf{A} \cdot \mathcal{O}[\![\mathsf{P}_2]\!]$, $\mathcal{O}[\![\mathsf{P}_1]\!] \neq \mathcal{O}[\![\mathsf{P}_2]\!]$ and this contradicts the hypothesis.

Otherwise $\mathcal{B}[\![\mathbf{G}' \ in \ \mathsf{P}_1]\!] \neq \mathcal{B}[\![\mathbf{G}' \ in \ \mathsf{P}_2]\!]$ and then, by inductive hypothesis, we have a contradiction. ■

Using the extension operator $\triangleright$ we can define a semantic operator $\uplus$ which computes the *OR*-composition of two denotations. Namely, given $\mathsf{D}_1, \mathsf{D}_2 \in \mathbb{PC}$,

$$\mathsf{D}_1 \uplus \mathsf{D}_2 := [\mathsf{D}_1 + \mathsf{D}_2]^* \tag{2.5.2}$$

where $[\mathsf{D}]^*$ is the least solution of the equation

$$[\mathsf{D}]^* = Id_{\mathbb{I}} + [\mathsf{D}]^* \triangleright su(\mathsf{D}), \tag{2.5.3}$$

or (equivalently) the least fixpoint of the operator

$$\mathcal{H}_{\mathsf{D}}(\mathsf{D}') := Id_{\mathbb{I}} + \mathsf{D}' \triangleright su(\mathsf{D}). \tag{2.5.4}$$

Theorem 2.5.9 shows the *OR-compositionality* property of $\mathcal{O}[\![\cdot]\!]$, i.e., the compositionality w.r.t. the $\cup$ operator. First we need the following (technical) lemma and its corollary.

Lemma 2.5.7 *Let* $\mathsf{D}, \mathsf{D}' \in \mathbb{PC}$. *Then* $su(\mathsf{D} \triangleright su(\mathsf{D}')) \sqsubseteq su(\mathsf{D}) \triangleright su(\mathsf{D}')$.

Proof. The following facts hold.

$$\begin{aligned}
& su(\mathsf{D} \triangleright su(\mathsf{D}')) = \\
& \qquad [\text{by definition of } su] \\
& \sum\{(\mathsf{A} \cdot (\mathsf{D} \triangleright su(\mathsf{D}'))) \times Id_{\mathbb{C}}\}_{\mathsf{A} \in Atoms} = \\
& \qquad [\text{by Point 1 of Lemma 2.5.1 and by definition of } Id_{\mathbb{C}}] \\
& \sum\{((\mathsf{A} \cdot \mathsf{D}) \triangleright su(\mathsf{D}')) \times \sum\{\phi_{\mathsf{G}}\}_{\mathsf{G} \in Goals}\}_{\mathsf{A} \in Atoms} = \\
& \qquad [\text{by Lemma 2.2.7}] \\
& \sum\{\sum\{((\mathsf{A} \cdot \mathsf{D}) \triangleright su(\mathsf{D}')) \times \phi_{\mathsf{G}}\}_{\mathsf{G} \in Goals}\}_{\mathsf{A} \in Atoms} \sqsubseteq \\
& \qquad [\text{by Point 2 of Lemma 2.5.1}] \\
& \sum\{\sum\{((\mathsf{A} \cdot \mathsf{D}) \times \phi_{\mathsf{G}}) \triangleright su(\mathsf{D}')\}_{\mathsf{G} \in Goals}\}_{\mathsf{A} \in Atoms} = \\
& \qquad [\text{by Lemma 2.2.7}]
\end{aligned}$$

$$\sum\{(\mathsf{A}\cdot\mathsf{D})\times\sum\{\phi_{\mathsf{G}}\}_{\mathsf{G}\in Goals}\}_{\mathsf{A}\in Atoms}\triangleright su(\mathsf{D}')=$$
$$[\text{by definition of } Id_{\mathbb{C}} \text{ and of } su]$$
$$su(\mathsf{D})\triangleright su(\mathsf{D}').$$

■

Corollary 2.5.8 *Let* $\mathsf{D},\mathsf{D}'\in\mathbb{PC}$. *Then, for any* $\mathsf{k}\geq 0$, $su(\mathsf{D}\triangleright su_{\mathsf{k}}(\mathsf{D}'))\sqsubseteq su(\mathsf{D})\triangleright su_{\mathsf{k}}(\mathsf{D}')$.

Proof. The proof is by induction on k. If $\mathsf{k}=0$, by definition of $\triangleright$ and since $su_0(\mathsf{D}')=\phi$, $su(\mathsf{D}\triangleright\phi)=su(\mathsf{D})=su(\mathsf{D})\triangleright\phi$.

Otherwise, note that, by definition of su_{k}, $su(\mathsf{D}\triangleright su_{\mathsf{k}}(\mathsf{D}'))=su(\mathsf{D}\triangleright(su_{\mathsf{k}-1}(\mathsf{D}')\triangleright su(\mathsf{D}')))$ and, for $\mathsf{k}>1$, $su(\mathsf{D}')\sqsubseteq su_{\mathsf{k}-1}(\mathsf{D}')$. Then by Lemma 2.2.6, $\mathsf{D}\triangleright su_{\mathsf{k}}(\mathsf{D}')=(\mathsf{D}\triangleright su_{\mathsf{k}-1}(\mathsf{D}'))\triangleright su(\mathsf{D}')$. This result trivially holds also for $\mathsf{k}=1$. To conclude

$$\begin{aligned}
&su(\mathsf{D}\triangleright su_{\mathsf{k}}(\mathsf{D}'))= && [\text{by the previous result}]\\
&su((\mathsf{D}\triangleright su_{\mathsf{k}-1}(\mathsf{D}'))\triangleright su(\mathsf{D}'))\sqsubseteq && [\text{by Lemma 2.5.7}]\\
&su(\mathsf{D}\triangleright su_{\mathsf{k}-1}(\mathsf{D}'))\triangleright su(\mathsf{D}')\sqsubseteq && [\text{by inductive hypothesis}]\\
&(su(\mathsf{D})\triangleright su_{\mathsf{k}-1}(\mathsf{D}'))\triangleright su(\mathsf{D}')= && [\text{by the previous result}]\\
&su(\mathsf{D})\triangleright su_{\mathsf{k}}(\mathsf{D}').
\end{aligned}$$

■

Theorem 2.5.9 *Let* P_1, P_2 *be programs. Then* $\mathcal{O}[\![\mathsf{P}_1\cup\mathsf{P}_2]\!]=\mathcal{O}[\![\mathsf{P}_1]\!]\uplus\mathcal{O}[\![\mathsf{P}_2]\!]$. *(OR-compositionality of the top-down denotation)*

Proof. By definition of $\uplus$, $\mathcal{O}[\![\mathsf{P}_1]\!]\uplus\mathcal{O}[\![\mathsf{P}_2]\!]=[\mathcal{O}[\![\mathsf{P}_1]\!]+\mathcal{O}[\![\mathsf{P}_2]\!]]^*$. Since $\mathbb{PC}$ is a complete lattice, $[\mathsf{D}]^*=lfp(\mathcal{H}_{\mathsf{D}})=\mathcal{H}_{\mathsf{D}}\uparrow\omega=\sum\{\mathcal{H}_{\mathsf{D}}\uparrow n\}_{n\geq 0}$, where $\mathcal{H}_{\mathsf{D}}:\mathbb{PC}\to\mathbb{PC}$ is the continuous function of equation (2.5.4). First of all, we prove by induction that, for any $n\geq 1$, $\mathcal{H}_{\mathsf{D}}\uparrow n=Id_{\mathbb{I}}\triangleright su_{n-1}(\mathsf{D})$. Then, by Lemma 2.2.7, $[\mathsf{D}]^*=Id_{\mathbb{I}}\triangleright\sum\{su_n(\mathsf{D})\}_{n\geq 0}$.

$\underline{n=1}$ $\mathcal{H}_{\mathsf{D}}\uparrow 1=\mathcal{H}_{\mathsf{D}}(\phi)=Id_{\mathbb{I}}+(\phi\triangleright su(\mathsf{D}))=Id_{\mathbb{I}}=Id_{\mathbb{I}}\triangleright su_0(\mathsf{D})$.

$\underline{n>1}$ The following hold.

$$\begin{aligned}
&\mathcal{H}_{\mathsf{D}}\uparrow n= && [\text{by definition of } \cdot\uparrow n]\\
&\mathcal{H}_{\mathsf{D}}(\mathcal{H}_{\mathsf{D}}\uparrow(n-1))= && [\text{by inductive hypothesis}]\\
&\mathcal{H}_{\mathsf{D}}(Id_{\mathbb{I}}\triangleright su_{n-1}(\mathsf{D}))= && [\text{by definition of } \mathcal{H}_{\mathsf{D}}]\\
&Id_{\mathbb{I}}+((Id_{\mathbb{I}}\triangleright su_{n-1}(\mathsf{D}))\triangleright su(\mathsf{D}))= && [\text{by Lemma 2.2.6}]\\
&Id_{\mathbb{I}}+(Id_{\mathbb{I}}\triangleright su_n(\mathsf{D}))= && [\text{since } \triangleright \text{ is extensive}]\\
&Id_{\mathbb{I}}\triangleright su_n(\mathsf{D}).
\end{aligned}$$

Now, to prove the thesis, we have to prove that

$$\sum\{su_n(tree(P_1 \cup P_2))\}_{n\geq 0} = \sum\{su_n(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!])\}_{n\geq 0}.$$

We prove the two inclusions separately.

$\sqsubseteq$ First of all observe that, since (for any program P) $tree(P)$ is a pure collection, $tree(P) \sqsubseteq Id_{\mathbb{I}} \triangleright su(tree(P)) \sqsubseteq \mathcal{O}[\![P]\!]$. Then, by definition of $tree$, $tree(P_1 \cup P_2) = tree(P_1) + tree(P_2) \sqsubseteq \mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!]$ and therefore, since $\cdot$ and $\times$ are monotonic, $su(tree(P_1 \cup P_2)) \sqsubseteq su(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!])$. Then, since $\triangleright$ is also monotonic, for any $n \geq 0$, $su_n(tree(P_1 \cup P_2)) \sqsubseteq su_n(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!])$.

$\sqsupseteq$ We prove (by induction on h) that, for any derivation d, if there exists $h \geq 0$ such that $d \in su_h(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!])(\mathbf{G})$ then there exists $k \geq 0$ such that $d \in su_k(tree(P_1 \cup P_2))(\mathbf{G})$. If $h = 0$ simply choose $k = 0$.

Otherwise let $h > 0$ and observe that, by definition of su_h and by Lemma 2.2.6,

$$su_h(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!]) = su_{h-1}(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!]) \triangleright su(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!]). \quad (1)$$

We have two possibilities. If $d \in su_{h-1}(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!])(\mathbf{G})$ then, by inductive hypothesis, there exists $k \geq 0$ such that $d \in su_k(tree(P_1 \cup P_2))(\mathbf{G})$ and then the thesis follows.

Otherwise, by definition of $\triangleright$, by (1) and since $su(\mathcal{O}[\![P_1]\!]+\mathcal{O}[\![P_2]\!])$ is closed under renaming, $d = d_1 :: d_2$, where $d_1 \in su_{h-1}(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!])(\mathbf{G})$, $last(d_1) = \mathbf{B}$ and

$$d_2 \in (su(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!]))(\mathbf{B}). \quad (2)$$

By inductive hypothesis, there exists $m \geq 0$ such that

$$d_1 \in su_m(tree(P_1 \cup P_2))(\mathbf{G}). \quad (3)$$

Now note that, by Lemma 2.2.7, $su(\mathcal{O}[\![P_1]\!] + \mathcal{O}[\![P_2]\!]) = su(\mathcal{O}[\![P_1]\!]) + su(\mathcal{O}[\![P_2]\!])$ and therefore, by (2), $d_2 \in (su(\mathcal{O}[\![P_1]\!]) + su(\mathcal{O}[\![P_2]\!]))(\mathbf{B})$. Now assume, without loss of generality, that $d_2 \in su(\mathcal{O}[\![P_1]\!])(\mathbf{B})$. Then, by Point 2 of Corollary 2.5.2, by Lemma 2.2.7 and since $tree(P_1) \sqsubseteq tree(P_1 \cup P_2)$, there exists $l \geq 0$ such that

$$d_2 \in su(Id_{\mathbb{I}} \triangleright su_l(tree(P_1)))(\mathbf{B}) \subseteq su(Id_{\mathbb{I}} \triangleright su_l(tree(P_1 \cup P_2)))(\mathbf{B}). \quad (4)$$

Moreover, by Corollary 2.5.8 and since $su(Id_{\mathbb{I}}) \sqsubseteq Id_{\mathbb{C}}$, $su(Id_{\mathbb{I}} \triangleright su_l(tree(P_1 \cup P_2))) \sqsubseteq su(Id_{\mathbb{I}}) \triangleright su_l(tree(P_1 \cup P_2)) \sqsubseteq Id_{\mathbb{C}} \triangleright su_l(tree(P_1 \cup P_2))$.

By the previous result and by (4), $\mathtt{d}_2 \in (Id_{\mathbb{C}} \triangleright su_{\mathfrak{l}}(tree(\mathsf{P}_1 \cup \mathsf{P}_2)))(\mathbf{B})$ and therefore, by (3) and since $\mathtt{d} = \mathtt{d}_1 :: \mathtt{d}_2$, $\mathtt{d} \in (su_{\mathfrak{m}}(tree(\mathsf{P}_1 \cup \mathsf{P}_2)) \triangleright (Id_{\mathbb{C}} \triangleright su_{\mathfrak{l}}(tree(\mathsf{P}_1 \cup \mathsf{P}_2))))(\mathbf{G})$. Finally note that, by definition of $\triangleright$, for any $\mathsf{D} \in \mathbb{C}$, $Id_{\mathbb{C}} \triangleright \mathsf{D} = Id_{\mathbb{C}} + \mathsf{D}$ and $\mathsf{D} \triangleright Id_{\mathbb{C}} = \mathsf{D}$. Then, by (2.2.7) and since $\triangleright$ is additive and extensive, $\mathtt{d} \in (su_{\mathfrak{m}}(tree(\mathsf{P}_1 \cup \mathsf{P}_2)) \triangleright su_{\mathfrak{l}}(tree(\mathsf{P}_1 \cup \mathsf{P}_2)))(\mathbf{G}) = su_{\mathfrak{m}+\mathfrak{l}}(tree(\mathsf{P}_1 \cup \mathsf{P}_2))(\mathbf{G})$. ■

In Theorem 2.5.15 we will prove that the top-down and the bottom-up denotations are equivalent, which implies (by Theorem 2.5.4) the equivalence between the denotational and the operational semantics.

In the following, to simplify the notation, given a pure collection D, we denote by $pu(\mathsf{D})$, $pu_{\mathfrak{n}}(\mathsf{D})$ and $pu^{\mathfrak{n}}(\mathsf{D})$ respectively the collections[9]

$$pu(\mathsf{D}) := \sum \left\{ \mathcal{G}[\![\mathbf{G}]\!]_{\mathsf{D}} \right\}_{\mathbf{G} \in Goals}, \tag{2.5.5}$$

$$pu_{\mathfrak{n}}(\mathsf{D}) := \underbrace{pu(\mathsf{D}) \triangleright \cdots \triangleright pu(\mathsf{D})}_{\mathfrak{n}}, \tag{2.5.6}$$

$$pu^{\mathfrak{n}}(\mathsf{D}) := \underbrace{pu(\mathsf{D} \triangleright pu(\mathsf{D} \triangleright pu(\cdots)))}_{\mathfrak{n}}. \tag{2.5.7}$$

Note that $pu(\mathsf{D})$ can be viewed as the *parallel unfolding* of the pure collection D and (analogously to $su(\mathsf{D})$) it is closed under renaming and under instantiation, since we consider all the possible evaluations of D. It is interesting to note that the operators su and pu enjoy some closure properties. Namely, given a pure collection D the following properties hold.

- If $\mathtt{d}$ is a renamed version of an element $\mathtt{d}' \in su(\mathsf{D})(\mathbf{G})$, by using a renaming ρ, then $\mathtt{d} \in su(\mathsf{D})(\mathbf{G}\rho)$. The same holds for pu.

- Using Lemma 2.2.4, it is easy to check that, for any idempotent substitution δ, goal $\mathbf{G}$ and derivation $\mathtt{d}$, such that $\partial_\delta(\mathtt{d})$ is defined, $\mathtt{d} \in su(\mathsf{D})(\mathbf{G})$ implies $\partial_\delta(\mathtt{d}) \in su(\mathsf{D})(\mathbf{G}\delta)$. Moreover, if $\mathtt{d} \in su(\mathsf{D})(\mathbf{G}\delta)$ and $var(\mathtt{d}) \cap var(\mathbf{G}) \subseteq var(\mathbf{G}\delta)$, then there exists a derivation $\mathtt{d}' \in su(\mathsf{D})(\mathbf{G})$ such that $clauses(\mathtt{d}') = clauses(\mathtt{d})$ and $\mathtt{d} = \partial_\delta(\mathtt{d}')$. The same for holds pu.

Using the definition of $pu(\mathsf{D})$ we can replace the equation (2.2.5) of $\mathcal{C}[\![\cdot]\!]$ by the equation $\mathcal{C}[\![\mathtt{c}]\!]_{\mathsf{I}} = tree(\mathtt{c}) \triangleright pu(\mathsf{I})$ and it is easy to check that the equation (2.2.4) can be replaced by the equation

$$\mathcal{P}[\![\mathsf{P}]\!]_{\mathsf{I}} = Id_{\mathbb{I}} + (tree(\mathsf{P}) \triangleright pu(\mathsf{I})). \tag{2.5.8}$$

[9] Note that $pu_1(\mathsf{D}) = pu^1(\mathsf{D}) = pu(\mathsf{D})$ and we assume that $pu_0(\mathsf{D}) := pu^0(\mathsf{D}) := \phi$.

The proof of the equivalence between the denotational and the operational semantics is mainly achieved by proving that the parallel unfolding can be simulated by the sequential one. Corollary 2.5.11 proves a form of associativity of the parallel unfolding which reverses from bottom-up to top-down. Then, Lemma 2.5.12 states that a step of sequential unfolding can be safely replaced by a step of parallel unfolding and that the parallel unfolding of a (finite) goal can be simulated by (a finite number of steps of) the sequential unfolding.

Lemma 2.5.10 *Let* $\mathsf{D}, \mathsf{D}' \in \mathbb{PC}$ *and* $\mathsf{h} \geq 0$. *Then*

1. $pu(\mathsf{D}) \triangleright pu(\mathsf{D}') \sqsubseteq pu(\mathsf{D} \triangleright pu(\mathsf{D}'))$.
2. $pu^{\mathsf{h}}(\mathsf{D}) \sqsubseteq \sum\{pu_{\mathsf{k}}(\mathsf{D})\}_{\mathsf{k}\geq 0}$.

Corollary 2.5.11 *Let* $\mathsf{D} \in \mathbb{PC}$. *Then* $\sum\{pu^{\mathsf{h}}(\mathsf{D})\}_{\mathsf{h}\geq 0} = \sum\{pu_{\mathsf{k}}(\mathsf{D})\}_{\mathsf{k}\geq 0}$.

Proof. The inclusion $\sqsubseteq$ is straightforward by Lemma 2.5.10. For the other inclusion we prove (by induction on n) that, for any $\mathsf{n} \geq 0$, $pu_{\mathsf{n}}(\mathsf{D}) \sqsubseteq pu^{\mathsf{n}}(\mathsf{D})$. If $\mathsf{n} = 0$, by definition, $pu^{0}(\mathsf{D}) = \phi = pu_{0}(\mathsf{D})$.

Otherwise the following holds.

$$\begin{array}{ll}
pu_{\mathsf{n}}(\mathsf{D}) = & [\text{by (2.5.6) and by (2.2.7)}] \\
pu(\mathsf{D}) \triangleright pu_{\mathsf{n}-1}(\mathsf{D}) \sqsubseteq & [\text{by inductive hypothesis and since } \triangleright \text{ is monotonic}] \\
pu(\mathsf{D}) \triangleright pu^{(\mathsf{n}-1)}(\mathsf{D}) \sqsubseteq & [\text{by Lemma 2.5.10}] \\
pu(\mathsf{D} \triangleright pu^{(\mathsf{n}-1)}(\mathsf{D})) = & [\text{by (2.5.7)}] \\
pu^{\mathsf{n}}(\mathsf{D}). &
\end{array}$$

■

Lemma 2.5.12 *Let* $\mathsf{D} \in \mathbb{PC}$. *Then*

1. $su(\mathsf{D}) \sqsubseteq pu(\mathsf{D} + Id_{\mathbb{I}})$.
2. $pu(\mathsf{D}) \sqsubseteq Id_{\mathbb{C}} + \sum\{su_{\mathsf{n}}(\mathsf{D})\}_{\mathsf{n}\geq 0}$.

Corollary 2.5.13 *Let* $\mathsf{D} \in \mathbb{PC}$. *Then*

$$\sum \left\{ pu^{\mathsf{h}}(Id_{\mathbb{I}} + \mathsf{D}) \right\}_{\mathsf{h}\geq 0} = Id_{\mathbb{C}} + \sum \left\{ su_{\mathsf{k}}(\mathsf{D}) \right\}_{\mathsf{k}\geq 0}.$$

Note that, by (2.5.5), by (2.3.2) and by definition of $\triangleright$, $+$ and $Id_{\mathbb{I}}$, for any $\mathsf{D} \in \mathbb{PC}$,

$$Id_{\mathbb{I}} + \mathsf{D} = Id_{\mathbb{I}} \triangleright pu(\mathsf{D}) = Id_{\mathbb{I}} \triangleright su(\mathsf{D}). \tag{2.5.9}$$

Corollary 2.5.14 *Let* P *be a program. Then*

$$\mathcal{F}[\![\mathsf{P}]\!] = (Id_{\mathbb{I}} + tree(\mathsf{P})) \triangleright \sum \left\{ pu^{n}(Id_{\mathbb{I}} + tree(\mathsf{P})) \right\}_{n \geq 0}.$$

Proof. First of all recall that $\mathcal{P}[\![\mathsf{P}]\!]$ is continuous and ϕ is the bottom of $\mathbb{C}$. We will prove (by induction on n) that, for any $n > 0$, $\mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_n = (Id_{\mathbb{I}} + tree(\mathsf{P})) \triangleright pu^{(n-1)}(Id_{\mathbb{I}} + tree(\mathsf{P}))$. Then, by definition of $\mathcal{F}[\![\cdot]\!]$, $\mathcal{F}[\![\mathsf{P}]\!] = \sum\{\mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_n\}_{n \geq 0} = \sum\{(Id_{\mathbb{I}} + tree(\mathsf{P})) \triangleright pu^{n}(Id_{\mathbb{I}} + tree(\mathsf{P}))\}_{n \geq 0}$ and then the thesis follows by Lemma 2.2.7.

$\underline{n = 1}$ By (2.5.8), $\mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_1 = Id_{\mathbb{I}} + (tree(\mathsf{P}) \triangleright pu(\phi)) = Id_{\mathbb{I}} + tree(\mathsf{P}) = (Id_{\mathbb{I}} + tree(\mathsf{P})) \triangleright pu^{0}(Id_{\mathbb{I}} + tree(\mathsf{P}))$.

$\underline{n > 1}$ The following holds.

$$\begin{aligned}
&\mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_n = \\
&\qquad [\text{since } \mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_{(n-1)} \sqsubseteq \mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_n] \\
&\mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_n + \mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_{(n-1)} = \\
&\qquad [\text{by definition of } \cdot{\uparrow}_n \text{ and } (2.5.8)] \\
&(Id_{\mathbb{I}} + (tree(\mathsf{P}) \triangleright pu(\mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_{(n-1)}))) + \mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_{(n-1)} = \\
&\qquad [\text{by } (2.5.9)] \\
&(tree(\mathsf{P}) \triangleright pu(\mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_{(n-1)})) + (Id_{\mathbb{I}} \triangleright pu(\mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_{(n-1)})) = \\
&\qquad [\text{by definition of } pu \text{ and Lemma 2.2.7 twice}] \\
&(Id_{\mathbb{I}} + tree(\mathsf{P})) \triangleright pu(\mathcal{P}[\![\mathsf{P}]\!]{\uparrow}_{(n-1)}) = \\
&\qquad [\text{by inductive hypothesis}] \\
&(Id_{\mathbb{I}} + tree(\mathsf{P})) \triangleright pu((Id_{\mathbb{I}} + tree(\mathsf{P})) \triangleright pu^{n-2}(Id_{\mathbb{I}} + tree(\mathsf{P}))) = \\
&\qquad [\text{by definition of } pu^{n}] \\
&(Id_{\mathbb{I}} + tree(\mathsf{P})) \triangleright pu^{(n-1)}(Id_{\mathbb{I}} + tree(\mathsf{P})).
\end{aligned}$$

■

Now we can prove the equivalence of the top-down and bottom-up goal-independent denotations.

Theorem 2.5.15 *Let* P *be a program. Then* $\mathcal{O}[\![\mathsf{P}]\!] = \mathcal{F}[\![\mathsf{P}]\!]$ *(equivalence of the top-down and bottom-up goal-independent denotations).*

Proof. By Corollaries 2.5.14 and 2.5.13, and since $\mathsf{D} \triangleright Id_{\mathbb{C}} = \mathsf{D}$,

$$\begin{aligned}
\mathcal{F}[\![\mathsf{P}]\!] &= (Id_{\mathbb{I}} + tree(\mathsf{P})) \triangleright \sum \{pu^{n}(Id_{\mathbb{I}} + tree(\mathsf{P}))\}_{n \geq 0} \\
&= (Id_{\mathbb{I}} + tree(\mathsf{P})) \triangleright \sum \{su_{n}(tree(\mathsf{P}))\}_{n \geq 0}.
\end{aligned} \tag{1}$$

Now, since $tree(\mathsf{P})$ is a pure collection and by (2.5.9), $Id_{\mathbb{I}} + tree(\mathsf{P}) = Id_{\mathbb{I}} \triangleright su(tree(\mathsf{P}))$. Finally, by Lemma 2.2.6, by (1), by Point 2 of Corollary 2.5.2, and by a straightforward inductive argumentation, $\mathcal{F}[\![\mathsf{P}]\!] = (Id_{\mathbb{I}} \triangleright su(tree(\mathsf{P}))) \triangleright \sum\{su_n(tree(\mathsf{P}))\}_{n\geq 0} = Id_{\mathbb{I}} \triangleright \sum\{su_n(tree(\mathsf{P}))\}_{n\geq 0} = \mathcal{O}[\![\mathsf{P}]\!]$. ■

Now we can show the *OR*-compositionality of the fixpoint denotation (which follows immediately from Theorems 2.5.15 and 2.5.9) and the equivalence between the denotational and the operational semantics.

Corollary 2.5.16 *Let* P_1, P_2 *be programs. Then* $\mathcal{F}[\![\mathsf{P}_1 \cup \mathsf{P}_2]\!] = \mathcal{F}[\![\mathsf{P}_1]\!] \uplus \mathcal{F}[\![\mathsf{P}_2]\!]$.

Corollary 2.5.17 *For any goal* $\mathbf{G}$ *and program* P, $\mathcal{Q}[\![\mathbf{G}\ in\ \mathsf{P}]\!] = \mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!]$. *(equivalence of the operational and denotational semantics)*

Proof. The proof is by structural induction on $\mathbf{G}$.

$\underline{\mathbf{G} = \square}$ By definition of $\mathcal{Q}[\![\cdot]\!]$, $\mathcal{F}[\![\cdot]\!]$ and $\mathcal{G}[\![\cdot]\!]$ and by Point 1 of Corollary 2.5.5, $\mathcal{Q}[\![\square\ in\ \mathsf{P}]\!] = \mathcal{G}[\![\square]\!]_{\mathcal{F}[\![\mathsf{P}]\!]} = \phi_{\square} = \mathcal{B}[\![\square\ in\ \mathsf{P}]\!]$.

$\underline{\mathbf{G} = (\mathbf{A}, \mathbf{G}')}$ The following equalities hold

$$
\begin{array}{ll}
\mathcal{Q}[\![(\mathsf{A}, \mathbf{G}')\ in\ \mathsf{P}]\!] = & [\text{by definition of } \mathcal{Q}[\![\cdot]\!] \text{ and } \mathcal{F}[\![\cdot]\!]] \\
\mathcal{G}[\![(\mathsf{A}, \mathbf{G}')]\!]_{\mathcal{F}[\![\mathsf{P}]\!]} = & [\text{by definition of } \mathcal{G}[\![\cdot]\!] \text{ and } \mathcal{Q}[\![\cdot]\!]] \\
\mathcal{A}[\![\mathsf{A}]\!]_{\mathcal{F}[\![\mathsf{P}]\!]} \times \mathcal{Q}[\![\mathbf{G}'\ in\ \mathsf{P}]\!] = & [\text{by definition of } \mathcal{A}[\![\cdot]\!] \text{ and induction}] \\
(\mathsf{A} \cdot \mathcal{F}[\![\mathsf{P}]\!]) \times \mathcal{B}[\![\mathbf{G}'\ in\ \mathsf{P}]\!] = & [\text{by Theorem 2.5.15}] \\
(\mathsf{A} \cdot \mathcal{O}[\![\mathsf{P}]\!]) \times \mathcal{B}[\![\mathbf{G}'\ in\ \mathsf{P}]\!] = & [\text{by Point 2 of Corollary 2.5.5}] \\
\mathcal{B}[\![(\mathsf{A}, \mathbf{G}')\ in\ \mathsf{P}]\!]. &
\end{array}
$$

■

2.6 Discussions on the Results

As already mentioned at the beginning of the chapter, our *SLD*-tree semantics was defined as the concrete (collecting) semantics of the framework presented in Chapter 3 for the systematic derivation of more abstract semantics, using the formal tools of abstract interpretation. The main novelty is in its definition style (denotational semantics and transition system), which has been conceived in such a way that it will be inherited by all the abstract semantics. This will also be the case for the compositionality and equivalence properties stated in Section 2.5.

Furthermore, the structure of the main results is performed keeping the abstraction framework in mind. Indeed the proof of all main theorems heavily rely on some

basic results, which express properties of the *abstract* primitive semantic operators. This technical lemmata embed (and "hide") *all* the properties of *SLD*-derivations which are needed to prove the main results. This is even more important since, by using the interaction between the properties of the *abstraction* and the properties of the *concrete* primitive semantic operators, we can easily inherit all those properties of the collecting semantics for which the suitable lemmata on the semantic operators hold. Then, when in the abstract case these lemmata will be proved, by using some properties of the abstraction, the abstract versions of the main results about collections will hold for free.

We will see that, due to the relation between the properties of the primitive semantic operators and the properties of the semantics, we can define a taxonomy of observables (abstractions). Each class in the taxonomy is characterized by a set of properties relating the primitive semantic operators and the Galois insertion which defines the observable. For each class we have

- a methodology to automatically derive the "best" abstract semantics (transition system, denotational semantics or both),
- the validity for the abstract semantics of some of the theorems which hold for the collecting semantics (equivalence between operational and denotational semantics, equivalence between top-down and bottom-up denotation, correctness, minimality and *AND* and *OR* compositionality).

The new relevant issue which can be discussed in the abstraction framework is *precision*, i.e., how good is the abstract semantics w.r.t. the abstraction of the collecting semantics. We have therefore classes of precise observables, where we can reconstruct all the semantics discussed in [13], and classes of approximate observables, where we can reconstruct several domains proposed for program analysis (groundness, types, ...). The abstraction framework has also been used as the semantic foundation of abstract diagnosis, which is presented in Chapter 5.

Let us finally note that, since our framework is based on standard operational and denotational semantic definitions, it can be adapted to other programming languages (especially extensions of logic programming).

2.7 Technical Proofs of the Chapter

In this section we give the technical proofs missing in the previous part of the chapter. We will need the following technical lemma.

Lemma 2.7.1 *Let* $\mathtt{d}_1, \mathtt{d}_2$ *be derivations and* σ, δ *be idempotent substitutions. Then the following holds.*

1. *If* $\sigma\delta$ *is idempotent and* $\partial_{\sigma\delta}(\mathsf{d}_1)$ *and* $\partial_\delta(\partial_\sigma(\mathsf{d}_1))$ *are defined, then* $\partial_{\sigma\delta}(\mathsf{d}_1) = \partial_\delta(\partial_\sigma(\mathsf{d}_1))$.

2. *If* $\partial_\sigma(\mathsf{d}_1 :: \mathsf{d}_2)$ *is defined, then either*

   - $length(\partial_\sigma(\mathsf{d}_1)) < length(\mathsf{d}_1)$ *and* $\partial_\sigma(\mathsf{d}_1 :: \mathsf{d}_2) = \partial_\sigma(\mathsf{d}_1)$ *or*
   - $length(\partial_\sigma(\mathsf{d}_1)) = length(\mathsf{d}_1)$ *and* $\partial_\sigma(\mathsf{d}_1 :: \mathsf{d}_2) = \partial_\sigma(\mathsf{d}_1) :: \partial_{\sigma'}(\mathsf{d}_2)$, *where* σ' *is an idempotent substitution such that* $last(\partial_\sigma(\mathsf{d}_1)) = (last(\mathsf{d}_1))\sigma'$.

Proof. The proof of Point 1 is straightforward by definition of ∂. To prove Point 2 observe that if $length(\partial_\sigma(\mathsf{d}_1)) < length(\mathsf{d}_1)$ then the proof is straightforward by definition of ∂ operation.

Otherwise, let $\mathbf{G} := (\mathsf{A}, \mathbf{G}')$, $\mathsf{c} := \mathsf{H} \leftarrow \mathbf{B}$ and σ be an idempotent substitution such that $var(\mathsf{c}) \cap var(\sigma) = \emptyset$. Moreover, let $\vartheta' := mgu(\mathsf{A}, \mathsf{H})$ and $\vartheta := mgu(\mathsf{A}\sigma, \mathsf{H})$. We prove that there exists an idempotent substitution σ' such that $(\mathbf{B}, \mathbf{G}')\vartheta'\sigma' = (\mathbf{B}, \mathbf{G}')\sigma\vartheta$. Then the proof follows by definition of derivation and by a straightforward inductive argument.

First of all observe that, since $var(\mathsf{c}) \cap var(\sigma) = \emptyset$, then $\vartheta = mgu(\mathsf{A}\sigma, \mathsf{H}\sigma)$ and therefore, by Lemma 1.3.3,

$$\sigma\vartheta = mgu(eqn(\sigma) \cup \{\mathsf{A} = \mathsf{H}\}) \tag{1}$$

Then, by Lemma 1.3.3, there exist $\vartheta'' := mgu(\mathsf{A}, \mathsf{H})$ and $\sigma'' := mgu(eqn(\sigma)\vartheta'')$ such that $\sigma\vartheta = \vartheta''\sigma''$. Moreover, by definition of mgu and since $\vartheta' = mgu(\mathsf{A}, \mathsf{H})$, there exists a renaming ρ such that $\vartheta'' = \vartheta'\rho$ and therefore, by (1), $\sigma\vartheta = \vartheta'\rho\sigma''$. Now let $\sigma' := (\rho\sigma'')|_{(\mathbf{B},\mathbf{G}')\vartheta'}$. Then

$$\begin{aligned}
&(\mathbf{B}, \mathbf{G}')\vartheta'\sigma' = && [\text{by definition of } \sigma'] \\
&(\mathbf{B}, \mathbf{G}')\vartheta'(\rho\sigma'')|_{(\mathbf{B},\mathbf{G}')\vartheta'} = && [\text{by definition of composition}] \\
&(\mathbf{B}, \mathbf{G}')\vartheta'\rho\sigma'' = && [\text{since } \sigma\vartheta = \vartheta'\rho\sigma''] \\
&(\mathbf{B}, \mathbf{G}')\sigma\vartheta.
\end{aligned}$$

Finally to prove the thesis we have only to prove that σ' is idempotent. First of all observe that, since ϑ' is idempotent, $dom(\vartheta') \cap var((\mathbf{B}, \mathbf{G}')\vartheta') = \emptyset$. Then, by definition of composition and since $dom(\sigma') \subseteq var((\mathbf{B}, \mathbf{G}')\vartheta')$, for any $\mathsf{x}/\mathsf{t} \in \sigma'$, $\mathsf{x}/\mathsf{t} \in \vartheta'\sigma'$ and therefore, since $\sigma' = (\rho\sigma'')|_{(\mathbf{B},\mathbf{G}')\vartheta'}$, $\mathsf{x}/\mathsf{t} \in \vartheta'\rho\sigma'' = \sigma\vartheta$. Then, the thesis follows since by construction $\sigma\vartheta$ is an idempotent substitution. ■

Now we can give the proof of all technical lemmata.

Proof of Lemma 2.2.3. The proof of Points 1 and 2 is straightforward by definition of $::$ and ∂ operation. To prove Point 3 assume that $\mathtt{d}_1 = \mathbf{G}'_0 \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \mathbf{G}'_k$ and let $\vartheta := \vartheta_1 \cdots \vartheta_k$ and $\mathbf{G}''_0 := \mathit{first}(\mathtt{d}_2)$ such that $\mathtt{d}_1 \wedge \mathtt{d}_2$ is defined. If $\mathbf{G}'_k \neq \Box$ then the thesis is straightforward by definition of $\wedge$.

Otherwise, by definition of $\wedge$, $\mathtt{d}_1 \wedge \mathtt{d}_2 = \mathtt{d}' :: \partial_\vartheta(\mathtt{d}_2)$, where $\mathtt{d}' = (\mathbf{G}'_0, \mathbf{G}''_0) \xrightarrow[c_1]{\vartheta_1} (\mathbf{G}'_1, \mathbf{G}''_0\vartheta_1) \xrightarrow[c_2]{\vartheta_2} \cdots \xrightarrow[c_k]{\vartheta_k} \mathbf{G}''_0\vartheta$ is a derivation. Moreover, since $\mathtt{d}_1 \wedge \mathtt{d}_2$ is defined, $\mathit{var}(\mathtt{d}_1) \cap \mathit{var}(\mathtt{d}_2) = \mathit{var}(\mathbf{G}'_0) \cap \mathit{var}(\mathbf{G}''_0)$ and therefore $\mathit{var}(\mathtt{d}_1) \cap \mathit{var}(\mathit{clauses}(\mathtt{d}_2)) = \emptyset$. Then, since $\mathit{var}(\mathbf{G}''_0\vartheta) \subseteq \mathit{var}(\mathbf{G}''_0) \cup \mathit{var}(\mathtt{d}_1)$, $\mathit{var}(\mathbf{G}''_0\vartheta) \cap \mathit{var}(\mathtt{d}_2) \subseteq \mathit{var}(\mathbf{G}''_0)$ and therefore $\partial_\vartheta(\mathtt{d}_2)$ is defined and, by Point 2, $\partial_\vartheta(\mathtt{d}_2)$ is a derivation of $\mathbf{G}''_0\vartheta$. Finally observe that by definition of ∂, $\mathit{var}(\mathtt{d}') \cap \mathit{var}(\partial_\vartheta(\mathtt{d}_2)) = \mathit{var}(\mathbf{G}''_0\vartheta)$. Therefore, $\mathtt{d}' :: \partial_\vartheta(\mathtt{d}_2)$ is defined and the thesis follows by Point 1. ■

Proof of Lemma 2.2.4. We prove the points separately.

Point 1 Let $\mathbf{G}_i := \mathit{first}(\mathtt{d}_i)$, for $i = 1, 2$. First of all observe that by definition of $\wedge$ it is easy to check that

$$\partial_\delta(\mathtt{d}_1 \wedge \mathbf{G}_2) = \partial_\delta(\mathtt{d}_1) \wedge \mathbf{G}_2\delta \text{ and} \tag{1}$$

$$\forall \beta \in \mathit{Subst}.\, \partial_\beta(\mathtt{d}_2) = \partial_{\beta|_{\mathbf{G}_2}}(\mathtt{d}_2). \tag{2}$$

Then in the following we can assume, without loss of generality, that given a derivation $\partial_\beta(\mathtt{d})$, $\mathit{dom}(\beta) \cap \mathit{var}(\mathtt{d}) \subseteq \mathit{var}(\mathit{first}(\mathtt{d}))$. We distinguish the following three cases.

last$(\mathtt{d}_1) \neq \Box$ In this case $\mathit{last}(\partial_\delta(\mathtt{d}_1)) \neq \Box$ and therefore, by definition of $\wedge$, $\mathtt{d}_1 \wedge \mathtt{d}_2 = \mathtt{d}_1 \wedge \mathbf{G}_2$ and $\partial_\delta(\mathtt{d}_1) \wedge \partial_\delta(\mathtt{d}_2) = \partial_\delta(\mathtt{d}_1) \wedge \mathbf{G}_2\delta$. Then the thesis follows by (1).

last$(\mathtt{d}_1) = \Box$ **and** ***last***$(\partial_\delta(\mathtt{d}_1)) \neq \Box$ Let $\mathtt{d}_1 := \mathbf{G}_1 \xrightarrow[c_1,\ldots,c_k]{\vartheta} \Box$. By definition of $\wedge$,

$$\mathtt{d}_1 \wedge \mathtt{d}_2 = (\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\vartheta(\mathtt{d}_2) \text{ and} \tag{3}$$

$$\partial_\delta(\mathtt{d}_1) \wedge \partial_\delta(\mathtt{d}_2) = \partial_\delta(\mathtt{d}_1) \wedge \mathbf{G}_2\delta. \tag{4}$$

Moreover, by the previous hypothesis, $\mathit{length}(\partial_\delta(\mathtt{d}_1)) < \mathit{length}(\mathtt{d}_1)$ and therefore $\mathit{length}(\partial_\delta(\mathtt{d}_1 \wedge \mathbf{G}_2)) < \mathit{length}(\mathtt{d}_1 \wedge \mathbf{G}_2)$. Then, by Point 2 of Lemma 2.7.1, $\partial_\delta((\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\vartheta(\mathtt{d}_2)) = \partial_\delta(\mathtt{d}_1 \wedge \mathbf{G}_2)$ and therefore

$$\begin{aligned}
&\partial_\delta(\mathtt{d}_1 \wedge \mathtt{d}_2) = && [\text{by (3) and last result}]\\
&\partial_\delta(\mathtt{d}_1 \wedge \mathbf{G}_2) = && [\text{by (1) and (4)}]\\
&\partial_\delta(\mathtt{d}_1) \wedge \partial_\delta(\mathtt{d}_2).
\end{aligned}$$

last$(\partial_\delta(\mathtt{d}_1)) = \square$ In this case $\mathit{last}(\mathtt{d}_1) = \square$ and then there exists $k \geq 0$ such that $\mathtt{d}_1 = \mathbf{G}_1 \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \square$ and $\partial_\delta(\mathtt{d}_1) = \mathbf{G}_1\delta \xrightarrow[c_1]{\sigma_1} \cdots \xrightarrow[c_k]{\sigma_k} \square$. Let $\vartheta := \vartheta_1 \cdots \vartheta_k$ and $\sigma := \sigma_1 \cdots \sigma_k$. Then, by definition of $\wedge$,

$$\mathtt{d}_1 \wedge \mathtt{d}_2 = (\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\vartheta(\mathtt{d}_2) \text{ and} \tag{5}$$
$$\partial_\delta(\mathtt{d}_1) \wedge \partial_\delta(\mathtt{d}_2) = (\partial_\delta(\mathtt{d}_1) \wedge \mathbf{G}_2\delta) :: \partial_\sigma(\partial_\delta(\mathtt{d}_2)). \tag{6}$$

Now observe that, since $\mathit{var}(\mathbf{G}_1) \cap \mathit{var}(\mathit{clauses}(\mathtt{d}_1)) = \emptyset$, by definition of derivation, $\delta\sigma$ is idempotent. Moreover, since $\mathtt{d}_1 \wedge \mathtt{d}_2$ and $\partial_\vartheta(\mathtt{d}_2)$ are defined, $(\mathit{var}(\delta) \cup \mathit{var}(\sigma)) \cap \mathit{var}(\mathit{clauses}(\mathtt{d}_2)) = \emptyset$ and $\partial_{\sigma\vartheta}(\mathtt{d}_2)$ is defined.

Since $\mathit{length}(\partial_\delta(\mathtt{d}_1)) = \mathit{length}(\mathtt{d}_1)$, by Point 2 of Lemma 2.7.1,

$$\partial_\delta((\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\vartheta(\mathtt{d}_2)) = \partial_\delta(\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\beta(\partial_\vartheta(\mathtt{d}_2)), \tag{7}$$

where $\mathbf{G}_2\delta\sigma = \mathit{last}(\partial_\delta(\mathtt{d}_1 \wedge \mathbf{G}_2)) = \mathit{last}(\mathtt{d}_1 \wedge \mathbf{G}_2)\beta = \mathbf{G}_2\vartheta\beta$. Then

$$(\delta\sigma)|_{\mathbf{G}_2} = (\vartheta\beta)|_{\mathbf{G}_2} \tag{8}$$

and, since $\delta\sigma$ is idempotent, $(\vartheta\beta)|_{\mathbf{G}_2}$ is also idempotent and, by (4), $\partial_{\vartheta\beta}(\mathtt{d}_2)$ is defined. Finally

$$\begin{aligned}
&\partial_\delta(\mathtt{d}_1 \wedge \mathtt{d}_2) = && [\text{by (5) and (7)}]\\
&\partial_\delta(\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\beta(\partial_\vartheta(\mathtt{d}_2)) = && [\text{by Point 1 of Lemma 2.7.1}]\\
&\partial_\delta(\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_{\vartheta\beta}(\mathtt{d}_2) = && [\text{by (8) and (2)}]\\
&\partial_\delta(\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_{\delta\sigma}(\mathtt{d}_2) = && [\text{by Point 1 of Lemma 2.7.1}]\\
&\partial_\delta(\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\sigma(\partial_\delta(\mathtt{d}_2)) = && [\text{by (1) and (6)}]\\
&\partial_\delta(\mathtt{d}_1) \wedge \partial_\delta(\mathtt{d}_2).
\end{aligned}$$

Point 2 We have two possibilities. If $\mathit{last}(\mathtt{d}_1) \neq \square$, $\mathtt{d}_1 \wedge (\mathtt{d}_2 \wedge \mathtt{d}_3) = \mathtt{d}_1 \wedge (\mathbf{G}_2, \mathbf{G}_3)$ and $(\mathtt{d}_1 \wedge \mathtt{d}_2) \wedge \mathtt{d}_3 = (\mathtt{d}_1 \wedge \mathbf{G}_2) \wedge \mathbf{G}_3$. Then the proof is straightforward by definition of $\wedge$.

Otherwise, let $\mathtt{d}_1 := \mathbf{G}_1 \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \square$ and $\vartheta := \vartheta_1 \cdots \vartheta_k$. By definition of $\wedge$,

$$\mathtt{d}_1 \wedge \mathtt{d}_2 = (\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\vartheta(\mathtt{d}_2) \tag{9}$$
$$\mathtt{d}_1 \wedge (\mathtt{d}_2 \wedge \mathtt{d}_3) = (\mathtt{d}_1 \wedge (\mathbf{G}_2, \mathbf{G}_3)) :: \partial_\vartheta(\mathtt{d}_2 \wedge \mathtt{d}_3) \tag{10}$$

Two cases arise.

$\underline{\boldsymbol{last}(\partial_\vartheta(\mathtt{d}_2)) \neq \square}$ In this case $\partial_\vartheta(\mathtt{d}_2) \wedge \partial_\vartheta(\mathtt{d}_3) = \partial_\vartheta(\mathtt{d}_2) \wedge \mathbf{G}_3\vartheta$ and therefore

$$
\begin{aligned}
&\mathtt{d}_1 \wedge (\mathtt{d}_2 \wedge \mathtt{d}_3) = && [\text{by (10) and Point 1}]\\
&(\mathtt{d}_1 \wedge (\mathbf{G}_2, \mathbf{G}_3)) :: (\partial_\vartheta(\mathtt{d}_2) \wedge \partial_\vartheta(\mathtt{d}_3)) = && [\text{by the previous observation}]\\
&(\mathtt{d}_1 \wedge (\mathbf{G}_2, \mathbf{G}_3)) :: (\partial_\vartheta(\mathtt{d}_2) \wedge \mathbf{G}_3\vartheta) = && [\text{by definition of } ::]\\
&((\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\vartheta(\mathtt{d}_2)) \wedge \mathbf{G}_3 = && [\text{since } \mathit{last}(\partial_\vartheta(\mathtt{d}_2)) \neq \square]\\
&((\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\vartheta(\mathtt{d}_2)) \wedge \mathtt{d}_3 = && [\text{by (9)}]\\
&(\mathtt{d}_1 \wedge \mathtt{d}_2) \wedge \mathtt{d}_3.
\end{aligned}
$$

$\underline{\boldsymbol{last}(\partial_\vartheta(\mathtt{d}_2)) = \square}$ Let $\partial_\vartheta(\mathtt{d}_2) := \mathbf{G}_2\vartheta \xrightarrow[c'_1]{\sigma_1} \cdots \xrightarrow[c'_m]{\sigma_m} \square$ and $\sigma := \sigma_1 \cdots \sigma_m$. Then, by definition of $\wedge$,

$$
\begin{aligned}
\partial_\vartheta(\mathtt{d}_2) \wedge \partial_\vartheta(\mathtt{d}_3) &= (\partial_\vartheta(\mathtt{d}_2) \wedge \mathbf{G}_3\vartheta) :: \partial_\sigma(\partial_\vartheta(\mathtt{d}_3))\\
&= \partial_\vartheta(\mathtt{d}_2 \wedge \mathbf{G}_3) :: \partial_\sigma(\partial_\vartheta(\mathtt{d}_3)).
\end{aligned} \tag{11}
$$

Moreover, observe that, by definition of $\wedge$,

$$
\mathtt{d}_1 \wedge \mathtt{d}_2 = (\mathbf{G}_1, \mathbf{G}_2) \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \mathbf{G}_2\vartheta \xrightarrow[c'_1]{\sigma_1} \cdots \xrightarrow[c'_m]{\sigma_m} \square \tag{12}
$$

and $\vartheta\sigma$ is an idempotent substitution. Furthermore, analogously to the the previous case, $\partial_{\vartheta\sigma}(\mathtt{d}_3)$ is defined. Finally

$$
\begin{aligned}
&\mathtt{d}_1 \wedge (\mathtt{d}_2 \wedge \mathtt{d}_3) =\\
&\qquad [\text{by (10), Point 1 and (11)}]\\
&(\mathtt{d}_1 \wedge (\mathbf{G}_2, \mathbf{G}_3)) :: (\partial_\vartheta(\mathtt{d}_2 \wedge \mathbf{G}_3) :: \partial_\sigma(\partial_\vartheta(\mathtt{d}_3))) =\\
&\qquad [\text{since } :: \text{ is associative}]\\
&((\mathtt{d}_1 \wedge (\mathbf{G}_2, \mathbf{G}_3)) :: \partial_\vartheta(\mathtt{d}_2 \wedge \mathbf{G}_3)) :: \partial_\sigma(\partial_\vartheta(\mathtt{d}_3)) =\\
&\qquad [\text{by definition of } \wedge]\\
&(((\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\vartheta(\mathtt{d}_2)) \wedge \mathbf{G}_3) :: \partial_\sigma(\partial_\vartheta(\mathtt{d}_3)) =\\
&\qquad [\text{by Point 1 of Lemma 2.7.1}]\\
&(((\mathtt{d}_1 \wedge \mathbf{G}_2) :: \partial_\vartheta(\mathtt{d}_2)) \wedge \mathbf{G}_3) :: \partial_{\vartheta\sigma}(\mathtt{d}_3) =\\
&\qquad [\text{by (9), by (12), by (10) and Point 1}]\\
&(\mathtt{d}_1 \wedge \mathtt{d}_2) \wedge \mathtt{d}_3.
\end{aligned}
$$

■

Proof of Lemma 2.5.1. We prove the points separately.

Point 1 We prove the two inclusions separately.

$\sqsupseteq$ Let $\mathtt{d} \in ((\mathsf{A} \cdot \mathsf{D}') \triangleright su(\mathsf{D}))(\mathsf{A})$. If $\mathtt{d} \in (\mathsf{A} \cdot \mathsf{D}')(\mathsf{A})$ then, since $\triangleright$ is extensive and $\cdot$ is monotonic, $\mathtt{d} \in (\mathsf{A} \cdot (\mathsf{D}' \triangleright su(\mathsf{D})))(\mathsf{A})$.

Otherwise, by definition of $\triangleright$ and since $su(\mathsf{D})$ is closed under renaming, there exist two derivations $\mathtt{d}_1 \in (\mathsf{A} \cdot \mathsf{D}')(\mathsf{A})$ and $\mathtt{d}_2 \in su(\mathsf{D})(\mathbf{G})$ such that $\mathbf{G} = \mathit{last}(\mathtt{d}_1)$,

$$\mathtt{d} = \mathtt{d}_1 :: \mathtt{d}_2 \text{ and } \mathit{var}(\mathtt{d}_1) \cap \mathit{var}(\mathtt{d}_2) = \mathit{var}(\mathbf{G}). \tag{1}$$

By definition of $\cdot$ there exists a derivation $\mathtt{d}_3$, which is a renamed apart (w.r.t. A) version of an element in $\mathsf{D}'(\mathsf{A}')$, for some atom $\mathsf{A}' \leq \mathsf{A}$, and there exists an idempotent substitution δ such that $\mathit{first}(\mathtt{d}_3)\delta = \mathsf{A}$ and

$$\mathtt{d}_1 = \partial_\delta(\mathtt{d}_3). \tag{2}$$

Without loss of generality, we can assume that

$$\mathit{var}(\mathit{first}(\mathtt{d}_3)) \cap \mathit{var}(\mathtt{d}_2) = \emptyset. \tag{3}$$

Moreover, since $\mathsf{D}'(\mathsf{A}')$ is a well-formed set of derivations, we can assume that

$$\mathit{length}(\mathtt{d}_3) = \mathit{length}(\mathtt{d}_1) = \mathit{length}(\partial_\delta(\mathtt{d}_3)). \tag{4}$$

Then, by Point 2 of Lemma 2.7.1, there exists an idempotent substitution δ' such that $\mathit{last}(\mathtt{d}_1) = \mathbf{G} = \mathbf{G}'\delta'$ where $\mathbf{G}' = \mathit{last}(\mathtt{d}_3)$. Moreover, by definition of ∂, by (1) and (3), $\mathit{var}(\mathtt{d}_3) \cap \mathit{var}(\mathtt{d}_2) = \emptyset$ and therefore $\mathit{var}(\mathbf{G}') \cap \mathit{var}(\mathtt{d}_2) = \emptyset$. Then, by properties of $su(\mathsf{D})$ and since $\mathtt{d}_2 \in su(\mathsf{D})(\mathbf{G}'\delta')$, there exists a derivation $\mathtt{d}_4 \in su(\mathsf{D})(\mathbf{G}')$ such that $\mathit{var}(\mathtt{d}_3) \cap \mathit{var}(\mathtt{d}_4) = \mathit{var}(\mathbf{G}')$ and

$$\partial_{\delta'}(\mathtt{d}_4) = \mathtt{d}_2. \tag{5}$$

Then, by definition of $\triangleright$ and by properties of $su(\mathsf{D})$, $\mathtt{d}_3 :: \mathtt{d}_4$ is a renamed apart (w.r.t. A) version of an element in $(\mathsf{D}' \triangleright su(\mathsf{D}))(\mathsf{A}')$. Moreover, by definition of ∂ and of δ, $\partial_\delta(\mathtt{d}_3 :: \mathtt{d}_4) \in (\mathsf{A} \cdot (\mathsf{D}' \triangleright su(\mathsf{D})))(\mathsf{A})$. Finally the following hold.

$$\begin{aligned}
&\partial_\delta(\mathtt{d}_3 :: \mathtt{d}_4) = && [\text{by Point 2 of Lemma 2.7.1 and by (4)}]\\
&\partial_\delta(\mathtt{d}_3) :: \partial_{\delta'}(\mathtt{d}_4) = && [\text{by (2), (5) and (1)}]\\
&\mathtt{d}.
\end{aligned}$$

$\sqsubseteq$ Let $\mathtt{d} \in (\mathsf{A} \cdot (\mathsf{D}' \triangleright su(\mathsf{D})))(\mathsf{A})$. If $\mathtt{d} \in (\mathsf{A} \cdot \mathsf{D}')(\mathsf{A})$ then, since (by extensivity) $\mathsf{A} \cdot \mathsf{D}' \sqsubseteq (\mathsf{A} \cdot \mathsf{D}') \triangleright su(\mathsf{D})$, $\mathtt{d} \in ((\mathsf{A} \cdot \mathsf{D}') \triangleright su(\mathsf{D}))(\mathsf{A})$.

Otherwise, by definition of $\triangleright$ and of $\cdot$, there exists a renamed apart (w.r.t. A) version $\mathtt{d}'$ of an element in $(\mathsf{D}' \triangleright \mathit{su}(\mathsf{D}))(\mathsf{A}')$, for some atom $\mathsf{A}' \leq \mathsf{A}$, and there exists an idempotent substitution δ such that $\mathsf{A} = \mathit{first}(\mathtt{d}')\delta$ and

$$\partial_\delta(\mathtt{d}') = \mathtt{d}. \tag{6}$$

Since $\mathtt{d}'$ is a renamed version of an element in $(\mathsf{D}' \triangleright \mathit{su}(\mathsf{D}))(\mathsf{A}')$ and $\mathtt{d} \notin (\mathsf{A} \cdot \mathsf{D}')(\mathsf{A})$,

$$\mathtt{d}' = \mathtt{d}'_1 :: \mathtt{d}'_2, \tag{7}$$

where $\mathtt{d}'_1$ is a renamed version of an element in $\mathsf{D}'(\mathsf{A}')$, $\mathbf{G}' = \mathit{last}(\mathtt{d}'_1)$ and, since $\mathit{su}(\mathsf{D})$ is closed under renaming, $\mathtt{d}'_2 \in \mathit{su}(\mathsf{D})(\mathbf{G}')$. Then, by definition of $\cdot$,

$$\partial_\delta(\mathtt{d}'_1) \in (\mathsf{A} \cdot \mathsf{D}')(\mathsf{A}). \tag{8}$$

Moreover, by (6), (7) and (8) and since (by hypothesis) $\mathtt{d} \notin (\mathsf{A} \cdot \mathsf{D}')(\mathsf{A})$, $\mathit{length}(\partial_\delta(\mathtt{d}')) > \mathit{length}(\partial_\delta(\mathtt{d}'_1))$ and then $\mathit{length}(\partial_\delta(\mathtt{d}'_1)) = \mathit{length}(\mathtt{d}'_1)$. Therefore, by Point 2 of Lemma 2.7.1,

$$\partial_\delta(\mathtt{d}'_1 :: \mathtt{d}'_2) = \partial_\delta(\mathtt{d}'_1) :: \partial_{\delta'}(\mathtt{d}'_2), \tag{9}$$

where δ' is a substitution such that $\mathit{last}(\partial_\delta(\mathtt{d}'_1)) = \mathbf{G}'\delta' = (\mathit{last}(\mathtt{d}'_1))\delta'$. Then, since $\mathtt{d}'_2 \in \mathit{su}(\mathsf{D})(\mathbf{G}')$, $\partial_{\delta'}(\mathtt{d}'_2)$ is defined and, by properties of $\mathit{su}(\mathsf{D})$, $\partial_{\delta'}(\mathtt{d}'_2) \in \mathit{su}(\mathsf{D})(\mathbf{G}'\delta')$. Therefore, by definition of $\triangleright$, (8) and (9), $\partial_\delta(\mathtt{d}'_1) :: \partial_{\delta'}(\mathtt{d}'_2) \in ((\mathsf{A} \cdot \mathsf{D}') \triangleright \mathit{su}(\mathsf{D}))(\mathsf{A})$.

Now to prove the thesis it is sufficient to observe that, by (9), (7) and (6), $\partial_\delta(\mathtt{d}'_1) :: \partial_{\delta'}(\mathtt{d}'_2) = \partial_\delta(\mathtt{d}'_1 :: \mathtt{d}'_2) = \partial_\delta(\mathtt{d}') = \mathtt{d}$.

Point 2 By definition of $\sqsubseteq$, we have to prove that, for any $\mathbf{G}' \in \mathit{Goals}$, $((\mathsf{D}' \triangleright \mathit{su}(\mathsf{D})) \times \phi_{\mathbf{G}})(\mathbf{G}') \subseteq ((\mathsf{D}' \times \phi_{\mathbf{G}}) \triangleright \mathit{su}(\mathsf{D}))(\mathbf{G}')$. Let $\mathtt{d} \in ((\mathsf{D}' \triangleright \mathit{su}(\mathsf{D})) \times \phi_{\mathbf{G}})(\mathbf{G}')$. Then, by definition of $\times$, $\mathbf{G}' = (\mathbf{G}_0, \mathbf{G})$ and there exists a renamed version $\mathtt{d}_1$ of an element in $(\mathsf{D}' \triangleright \mathit{su}(\mathsf{D}))(\mathbf{G}_0)$ such that $\mathit{first}(\mathtt{d}_1) = \mathbf{G}_0$ and

$$\mathtt{d} = \mathtt{d}_1 \wedge \mathbf{G}. \tag{10}$$

Now, by definition of $\triangleright$, two cases arise. If $\mathtt{d}_1$ is a renamed version of an element in $\mathsf{D}'(\mathbf{G}_0)$ then, by definition of $\times$ and by (10), $\mathtt{d} \in (\mathsf{D}' \times \phi_{\mathbf{G}})(\mathbf{G}')$ and therefore, since $\triangleright$ is extensive, $\mathtt{d} \in ((\mathsf{D}' \times \phi_{\mathbf{G}}) \triangleright \mathit{su}(\mathsf{D}))(\mathbf{G}')$.

Otherwise, by definition of $\triangleright$ and since $\mathit{su}(\mathsf{D})$ is closed under renaming, $\mathtt{d}_1 = \mathtt{d}_3 :: \mathtt{d}_4$, where $\mathtt{d}_3 = \mathbf{G}_0 \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \mathbf{B}$ is a renamed version of an element in $\mathsf{D}'(\mathbf{G}_0)$,

$\mathbf{B} \neq \Box$, $\mathtt{d}_4 \in su(\mathsf{D})(\mathbf{B})$ and $var(\mathtt{d}_3) \cap var(\mathtt{d}_4) = var(\mathbf{B})$. Let $\vartheta := \vartheta_1 \cdots \vartheta_k$. By (10) and by definition of $\wedge$, $\mathtt{d} = (\mathtt{d}_3 \wedge \mathbf{G}) :: (\mathtt{d}_4 \wedge \mathbf{G}\vartheta)$ and, by definition of $\times$, $\mathtt{d}_3 \wedge \mathbf{G} \in (\mathsf{D}' \times \phi_{\mathbf{G}})(\mathbf{G}')$. Moreover, since $\mathtt{d}_4 \in su(\mathsf{D})(\mathbf{B})$, by definition of su and since $\mathbf{B} \neq \Box$, $\mathtt{d}_4 \wedge \mathbf{G}\vartheta \in su(\mathsf{D})(\mathbf{B}, \mathbf{G}\vartheta)$. Finally, by definition of $\triangleright$, $\mathtt{d} \in ((\mathsf{D}' \times \phi_{\mathbf{G}}) \triangleright su(\mathsf{D}))(\mathbf{G}')$.

Point 3 We prove the two inclusions separately.

$\sqsubseteq$ Let $\mathtt{d} \in ((\mathsf{D}' \times \mathsf{D}'') \triangleright su(\mathsf{D}))(\mathbf{G}_0)$. We have two possibilities. If $\mathtt{d} \in (\mathsf{D}' \times \mathsf{D}'')(\mathbf{G}_0)$ then, since $\triangleright$ is extensive and $\times$ is monotonic, $\mathtt{d} \in (\mathsf{D}' \times (\mathsf{D}'' \triangleright su(\mathsf{D})))(\mathbf{G}_0)$.

Otherwise, by definition of $\triangleright$ and since $su(\mathsf{D})$ is closed under renaming, there exist two derivations $\mathtt{d}' \in (\mathsf{D}' \times \mathsf{D}'')(\mathbf{G}_0)$ and $\mathtt{d}'' \in su(\mathsf{D})(\mathbf{G})$ such that $last(\mathtt{d}') = \mathbf{G}$,

$$\mathtt{d} = \mathtt{d}' :: \mathtt{d}'' \text{ and } var(\mathtt{d}') \cap var(\mathtt{d}'') = var(\mathbf{G}). \tag{11}$$

Then, by definition of $\times$, there exist two goals $\mathbf{G}'_0$, $\mathbf{G}''_0$ and two derivations $\mathtt{d}_2$ and $\mathtt{d}_1 = \mathbf{G}'_0 \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \mathbf{G}'_k$ (which are renamed versions of elements in $\mathsf{D}''(\mathbf{G}''_0)$ and $\mathsf{D}'(\mathbf{G}'_0)$ respectively) such that $first(\mathtt{d}_2) = \mathbf{G}''_0$, $\mathbf{G}_0 = (\mathbf{G}'_0, \mathbf{G}''_0)$ and $\mathtt{d}' = \mathtt{d}_1 \wedge \mathtt{d}_2$. Let $\vartheta := \vartheta_1 \cdots \vartheta_k$. Two cases arise

$\mathbf{G}'_k \neq \Box$ By definition of $\wedge$, $\mathtt{d}' = \mathtt{d}_1 \wedge \mathbf{G}''_0$. Since (by hypothesis) $\mathtt{d}_2$ is a renamed version of an element in $\mathsf{D}''(\mathbf{G}''_0)$, $\mathsf{D}''(\mathbf{G}''_0) \neq \emptyset$ and since (by definition of collection) $\mathsf{D}''(\mathbf{G}''_0)$ is a well-formed set of derivations, $\mathbf{G}''_0 \in \mathsf{D}''(\mathbf{G}''_0)$. By (11), since $last(\mathtt{d}_1) = \mathbf{G}'_k \neq \Box$ and $last(\mathtt{d}') = (\mathbf{G}'_k, \mathbf{G}''_0\vartheta)$,

$$\mathtt{d} = (\mathtt{d}_1 :: \mathtt{d}_3) \wedge \mathbf{G}''_0, \text{ where} \tag{12}$$

$$\mathtt{d}_3 \in su(\mathsf{D})(\mathbf{G}'_k) \text{ is such that } \mathtt{d}'' = \mathtt{d}_3 \wedge \mathbf{G}''_0\vartheta. \tag{13}$$

By definition of $\triangleright$, since $\mathtt{d}_1 :: \mathtt{d}_3$ is defined, since $\mathtt{d}_1$ is a renamed version of an element in $\mathsf{D}'(\mathbf{G}'_0)$ and by (13), $\mathtt{d}_1 :: \mathtt{d}_3$ is a renamed version of an element in $(\mathsf{D}' \triangleright su(\mathsf{D}))(\mathbf{G}'_0) = \mathsf{D}'(\mathbf{G}'_0)$. The last equality holds since (by hypothesis) $\mathsf{D}' \triangleright su(\mathsf{D}) = \mathsf{D}'$. Therefore, by (12), by definition of $\times$ and since $\mathbf{G}''_0 \in \mathsf{D}''(\mathbf{G}''_0)$, $\mathtt{d} = (\mathtt{d}_1 :: \mathtt{d}_3) \wedge \mathbf{G}''_0 \in (\mathsf{D}' \times \mathsf{D}'')(\mathbf{G}_0)$ and this contradicts the hypothesis.

$\mathbf{G}'_k = \Box$ By definition of $\wedge$,

$$\mathtt{d}' = \mathtt{d}_1 \wedge \mathtt{d}_2 = (\mathtt{d}_1 \wedge \mathbf{G}''_0) :: \partial_\vartheta(\mathtt{d}_2). \tag{14}$$

We can assume that $length(\mathtt{d}_2) = length(\partial_\vartheta(\mathtt{d}_2))$, since $\mathsf{D}''(\mathbf{G}''_0)$ is a well-formed set of derivations. Then, by definition of ∂, by (11), since (by

(14)) $last(\partial_\vartheta(\mathtt{d}_2)) = last(\mathtt{d}') = \mathbf{G}$ and by Point 2 of Lemma 2.7.1, there exists an idempotent substitution δ such that $last(\partial_\vartheta(\mathtt{d}_2)) = \mathbf{G}''\delta = \mathbf{G}$, where $\mathbf{G}'' = last(\mathtt{d}_2)$.

Note that by (11) and since $var(\mathtt{d}') = var(\mathtt{d}_1) \cup var(\mathtt{d}_2)$, $var(\mathtt{d}_2) \cap var(clauses(\mathtt{d}'')) = \emptyset$ and $var(\mathbf{G}'') \cap var(clauses(\mathtt{d}'')) = \emptyset$. Then, by properties of $su(\mathsf{D})$, since $\mathtt{d}'' \in su(\mathsf{D})(\mathbf{G})$ and $\mathbf{G} = \mathbf{G}''\delta$, there exists $\mathtt{d}_3 \in su(\mathsf{D})(\mathbf{G}'')$ such that

$$\mathtt{d}'' = \partial_\delta(\mathtt{d}_3) \text{ and } var(\mathtt{d}_3) \cap var(\mathtt{d}_2) = var(\mathbf{G}''). \tag{15}$$

Then $\mathtt{d}_2 :: \mathtt{d}_3$ is defined and, by Point 2 of Lemma 2.7.1 and since (by assumption) $length(\mathtt{d}_2) = length(\partial_\vartheta(\mathtt{d}_2))$,

$$\partial_\vartheta(\mathtt{d}_2 :: \mathtt{d}_3) = \partial_\vartheta(\mathtt{d}_2) :: \partial_\delta(\mathtt{d}_3). \tag{16}$$

Then

$$\begin{array}{ll}
\mathtt{d} = & [\text{by (11), (14) and (15)}] \\
(\mathtt{d}_1 \wedge \mathtt{d}_2) :: \partial_\delta(\mathtt{d}_3) = & [\text{by definition of } \wedge] \\
((\mathtt{d}_1 \wedge \mathbf{G}''_0) :: \partial_\vartheta(\mathtt{d}_2)) :: \partial_\delta(\mathtt{d}_3) = & [\text{since } :: \text{ is associative}] \\
(\mathtt{d}_1 \wedge \mathbf{G}''_0) :: (\partial_\vartheta(\mathtt{d}_2) :: \partial_\delta(\mathtt{d}_3)) = & [\text{by (16)}] \\
(\mathtt{d}_1 \wedge \mathbf{G}''_0) :: \partial_\vartheta(\mathtt{d}_2 :: \mathtt{d}_3) = & [\text{by definition of } \wedge \text{ and of } \mathtt{d}_1] \\
\mathtt{d}_1 \wedge (\mathtt{d}_2 :: \mathtt{d}_3). &
\end{array}$$

To conclude, by definition of $\triangleright$, since $\mathtt{d}_2$ is a renamed version of an element in $\mathsf{D}''(\mathbf{G}''_0)$, since $\mathtt{d}_3 \in su(\mathsf{D})(\mathbf{G}'')$ and $su(\mathsf{D})$ is closed under renaming, $\mathtt{d}_2 :: \mathtt{d}_3$ is a renamed version of an element in $(\mathsf{D}'' \triangleright su(\mathsf{D}))(\mathbf{G}''_0)$ such that $first(\mathtt{d}_2 :: \mathtt{d}_3) = \mathbf{G}''_0$. Hence, since $\mathtt{d}_1$ is a renamed version of an element in $\mathsf{D}'(\mathbf{G}'_0)$ and $first(\mathtt{d}_1) = \mathbf{G}'_0$, by definition of $\times$, $\mathtt{d}_1 \wedge (\mathtt{d}_2 :: \mathtt{d}_3) \in (\mathsf{D}' \times (\mathsf{D}'' \triangleright su(\mathsf{D})))(\mathbf{G}'_0, \mathbf{G}''_0)$.

$\sqsupseteq$ Let $\mathtt{d} \in (\mathsf{D}' \times (\mathsf{D}'' \triangleright su(\mathsf{D})))(\mathbf{G}_0)$. Then, by definition of $\times$, $\mathtt{d} = \mathtt{d}' \wedge \mathtt{d}''$, where $\mathbf{G}_0 = (\mathbf{G}'_0, \mathbf{G}''_0)$ and $\mathtt{d}'$, $\mathtt{d}''$ are renamed versions of elements in $\mathsf{D}'(\mathbf{G}'_0)$ and in $(\mathsf{D}'' \triangleright su(\mathsf{D}))(\mathbf{G}''_0)$ respectively, such that $first(\mathtt{d}') = \mathbf{G}'_0$ and $first(\mathtt{d}'') = \mathbf{G}''_0$. By definition of $\times$, two cases arise.

$\underline{\boldsymbol{last}(\mathtt{d}') \neq \square}$ In this case $\mathtt{d} = \mathtt{d}' \wedge \mathbf{G}''_0 \in (\mathsf{D}' \times \mathsf{D}'')(\mathbf{G}_0)$. Therefore, since $\triangleright$ is extensive, $\mathtt{d} \in ((\mathsf{D}' \times \mathsf{D}'') \triangleright su(\mathsf{D}))(\mathbf{G}_0)$.

$\underline{\boldsymbol{last}(\mathtt{d}') = \square}$ By definition of $\triangleright$, we distinguish two cases. If $\mathtt{d}''$ is a renamed version of an element in $\mathsf{D}''(\mathbf{G}''_0)$, $\mathtt{d} = \mathtt{d}' \wedge \mathtt{d}'' \in (\mathsf{D}' \times \mathsf{D}'')(\mathbf{G}_0)$ and therefore, analogously to the previous case, $\mathtt{d} \in ((\mathsf{D}' \times \mathsf{D}'') \triangleright su(\mathsf{D}))(\mathbf{G}_0)$.

Otherwise assume that $\mathtt{d}' = \mathbf{G}'_0 \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \square$ and let $\vartheta := \vartheta_1 \cdots \vartheta_k$. By definition of $\triangleright$ and since $su(\mathsf{D})$ is closed under renaming, $\mathtt{d}'' = (\mathtt{d}_1 :: \mathtt{d}_2)$, where $\mathtt{d}_1 = \mathbf{G}''_0 \xrightarrow[c'_1]{\sigma_1} \cdots \xrightarrow[c'_h]{\sigma_h} \mathbf{G}''_h$ is a renamed version of an element in $\mathsf{D}''(\mathbf{G}_0)$ and $\mathtt{d}_2 \in su(\mathsf{D})(\mathbf{G}''_h)$. Moreover, by definition of $\wedge$,

$$\mathtt{d} = (\mathtt{d}' \wedge \mathbf{G}''_0) :: \partial_\vartheta(\mathtt{d}'') = (\mathtt{d}' \wedge \mathbf{G}''_0) :: \partial_\vartheta(\mathtt{d}_1 :: \mathtt{d}_2). \tag{17}$$

Now we have two possibilities.

$\underline{\boldsymbol{length}(\partial_\vartheta(\mathtt{d}_1)) < \boldsymbol{length}(\mathtt{d}_1)}$ By Point 2 of Lemma 2.7.1, $\partial_\vartheta(\mathtt{d}_1 :: \mathtt{d}_2) = \partial_\vartheta(\mathtt{d}_1)$ and therefore, by (17) and by definition of $\wedge$, $\mathtt{d} = (\mathtt{d}' \wedge \mathbf{G}''_0) :: \partial_\vartheta(\mathtt{d}_1) = \mathtt{d}' \wedge \mathtt{d}_1 \in (\mathsf{D}' \times \mathsf{D}'')(\mathbf{G}_0) \subseteq ((\mathsf{D}' \times \mathsf{D}'') \triangleright su(\mathsf{D}))(\mathbf{G}_0)$.

$\underline{\boldsymbol{length}(\partial_\vartheta(\mathtt{d}_1)) = \boldsymbol{length}(\mathtt{d}_1)}$ By Point 2 of Lemma 2.7.1 there exists an idempotent substitution δ such that $\mathbf{G}''_h\delta = last(\partial_\vartheta(\mathtt{d}_1))$ and

$$\partial_\vartheta(\mathtt{d}_1 :: \mathtt{d}_2) = \partial_\vartheta(\mathtt{d}_1) :: \partial_\delta(\mathtt{d}_2). \tag{18}$$

Moreover, since $\partial_\delta(\mathtt{d}_2)$ is defined, by properties of $su(\mathsf{D})$ and since $\mathtt{d}_2 \in su(\mathsf{D})(\mathbf{G}''_h)$,

$$\partial_\delta(\mathtt{d}_2) \in su(\mathsf{D})(\mathbf{G}''_h\delta). \tag{19}$$

Then the following facts hold.

$$\begin{array}{ll}
\mathtt{d} = & [\text{by (17) and (18)}] \\
(\mathtt{d}' \wedge \mathbf{G}''_0) :: (\partial_\vartheta(\mathtt{d}_1) :: \partial_\delta(\mathtt{d}_2)) = & [\text{since :: is associative}] \\
((\mathtt{d}' \wedge \mathbf{G}''_0) :: \partial_\vartheta(\mathtt{d}_1)) :: \partial_\delta(\mathtt{d}_2) = & [\text{by definition of } \wedge] \\
(\mathtt{d}' \wedge \mathtt{d}_1) :: \partial_\delta(\mathtt{d}_2). &
\end{array}$$

Finally, by construction and (19), $(\mathtt{d}' \wedge \mathtt{d}_1) \in (\mathsf{D}' \times \mathsf{D}'')(\mathbf{G}_0)$, $\partial_\delta(\mathtt{d}_2) \in su(\mathsf{D})(\mathbf{G}''_h\delta)$. Therefore, since $(\mathtt{d}' \wedge \mathtt{d}_1) :: \partial_\delta(\mathtt{d}_2)$ is defined, by definition of $\triangleright$ and by the previous result, $\mathtt{d} = (\mathtt{d}' \wedge \mathtt{d}_1) :: \partial_\delta(\mathtt{d}_2) \in ((\mathsf{D}' \times \mathsf{D}'') \triangleright su(\mathsf{D}))(\mathbf{G}_0)$. ■

We need now another technical lemma.

Lemma 2.7.2 *Let* $\mathsf{D} \in \mathbb{PC}$, $\mathsf{D}_1, \mathsf{D}_2 \in \mathbb{C}$ *and* $\mathsf{A} \in \mathit{Atoms}$. *Then*

1. $(\mathsf{A} \cdot \mathsf{D}_1) \triangleright pu(\mathsf{D}) = \mathsf{A} \cdot (\mathsf{D}_1 \triangleright pu(\mathsf{D}))$.
2. $(\mathsf{D}_1 \times \mathsf{D}_2) \triangleright pu(\mathsf{D}) \sqsubseteq (\mathsf{D}_1 \triangleright pu(\mathsf{D})) \times (\mathsf{D}_2 \triangleright pu(\mathsf{D}))$.

Proof. The proof of Point 1 is analogous to the one of Point 1 of Lemma 2.5.1 and hence omitted.

To prove Point 2, let $\mathtt{d} \in ((\mathsf{D}_1 \times \mathsf{D}_2) \triangleright pu(\mathsf{D}))(\mathbf{G})$. If $\mathtt{d} \in (\mathsf{D}_1 \times \mathsf{D}_2)(\mathbf{G})$ then, since $\triangleright$ is extensive and $\times$ is monotonic, $\mathtt{d} \in ((\mathsf{D}_1 \triangleright pu(\mathsf{D})) \times (\mathsf{D}_2 \triangleright pu(\mathsf{D})))(\mathbf{G})$.

Otherwise, by definition of $\triangleright$ and since $pu(\mathsf{D})$ is closed under renaming,

$$\mathtt{d} = \mathtt{d}' :: \mathtt{d}'', \tag{1}$$

where $\mathtt{d}' \in (\mathsf{D}_1 \times \mathsf{D}_2)(\mathbf{G})$, $\mathit{last}(\mathtt{d}') = \mathbf{B} \neq \square$, $\mathtt{d}'' \in pu(\mathsf{D})(\mathbf{B})$ and

$$\mathit{var}(\mathtt{d}') \cap \mathit{var}(\mathtt{d}'') = \mathit{var}(\mathbf{B}). \tag{2}$$

By definition of $\times$, $\mathtt{d}' = \mathtt{d}_1 \wedge \mathtt{d}_2$, where $\mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2)$ and (for $\mathtt{i} = 1, 2$) $\mathtt{d}_\mathtt{i}$ is a renamed version of an element in $\mathsf{D}_\mathtt{i}(\mathbf{G}_\mathtt{i})$ such that $\mathit{first}(\mathtt{d}_\mathtt{i}) = \mathbf{G}_\mathtt{i}$ and $\mathit{var}(\mathtt{d}_1) \cap \mathit{var}(\mathtt{d}_2) = \mathit{var}(\mathbf{G}_1) \cap \mathit{var}(\mathbf{G}_2)$. Let $\mathtt{d}_1 = \mathbf{G}_1 \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \mathbf{B}'$ and $\vartheta := \vartheta_1 \cdots \vartheta_k$. Two cases arise but we prove only the case $\mathbf{B}' \neq \square$ since the other is analogous. First of all,

$$\mathtt{d}' = \mathtt{d}_1 \wedge \mathbf{G}_2, \tag{3}$$

$\mathbf{B} = (\mathbf{B}', \mathbf{G}_2\vartheta)$ and $\mathtt{d}'' \in pu(\mathsf{D})(\mathbf{B}', \mathbf{G}_2\vartheta)$. Moreover, by (2.5.5) and since (by Lemma 2.2.6) $\times$ is associative,

$$\mathtt{d}'' = \mathtt{d}_3 \wedge \mathtt{d}_4, \tag{4}$$

where $\mathtt{d}_3 \in pu(\mathsf{D})(\mathbf{B}')$ and $\mathtt{d}_4 \in pu(\mathsf{D})(\mathbf{G}_2\vartheta)$. By (2), $\mathit{var}(\mathtt{d}_1) \cap \mathit{var}(\mathtt{d}_3) = \mathit{var}(\mathbf{B}')$ and therefore, by definition of $\triangleright$ and since pu is closed under renaming,

$$\mathtt{d}_1 :: \mathtt{d}_3 \text{ is a renamed version of an element in } (\mathsf{D}_1 \triangleright pu(\mathsf{D}))(\mathbf{G}_1). \tag{5}$$

Moreover, since $\mathtt{d}_4 \in pu(\mathsf{D})(\mathbf{G}_2\vartheta)$, since (by our hypothesis on variables) $\mathit{var}(\mathtt{d}_4) \cap \mathit{var}(\mathbf{G}_2) \subseteq \mathit{var}(\mathbf{G}_2\vartheta)$ and by properties of $pu(\mathsf{D})$, there exists

$$\mathtt{d}_5 \in pu(\mathsf{D})(\mathbf{G}_2) \text{ such that } \mathtt{d}_4 = \partial_\vartheta(\mathtt{d}_5). \tag{6}$$

Now observe that, since $\mathsf{D}_2(\mathbf{G}_2) \neq \emptyset$ is a well-formed set of derivations, $\mathbf{G}_2 \in \mathsf{D}_2(\mathbf{G}_2)$ and therefore, by definition of $\triangleright$,

$$\mathtt{d}_5 = \mathbf{G}_2 :: \mathtt{d}_5 \in (\mathsf{D}_2 \triangleright pu(\mathsf{D}))(\mathbf{G}_2). \tag{7}$$

By our hypothesis on variables, $\mathit{var}(\mathtt{d}_1 :: \mathtt{d}_3) \cap \mathit{var}(\mathtt{d}_5) = \mathit{var}(\mathbf{G}_1) \cap \mathit{var}(\mathbf{G}_2)$. Therefore, by definition of $\times$, by (5), by (7) and since $\mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2)$, $(\mathtt{d}_1 :: \mathtt{d}_3) \wedge \mathtt{d}_5 \in ((\mathsf{D}_1 \triangleright pu(\mathsf{D})) \times (\mathsf{D}_2 \triangleright pu(\mathsf{D})))(\mathbf{G})$. Finally, since (by hypothesis) $\mathit{last}(\mathtt{d}_1) \neq \square$,

$$(\mathtt{d}_1 :: \mathtt{d}_3) \wedge \mathtt{d}_5 = \qquad [\text{by definition of } \wedge]$$

$$
\begin{aligned}
&(\mathtt{d}_1 \wedge \mathbf{G}_2) :: (\mathtt{d}_3 \wedge \partial_\vartheta(\mathtt{d}_5)) = && [\text{by } (6)]\\
&(\mathtt{d}_1 \wedge \mathbf{G}_2) :: (\mathtt{d}_3 \wedge \mathtt{d}_4) = && [\text{by } (3) \text{ and } (4)]\\
&\mathtt{d}' :: \mathtt{d}'' = && [\text{by } (1)]\\
&\mathtt{d}.
\end{aligned}
$$

■

Proof of Lemma 2.5.10. We prove the points separately.

Point 1 We have to prove that $\forall \mathbf{G} \in \mathit{Goals}.\,(pu(\mathsf{D}) \triangleright pu(\mathsf{D}'))(\mathbf{G}) \subseteq (pu(\mathsf{D} \triangleright pu(\mathsf{D}')))(\mathbf{G})$. The proof is by structural induction on $\mathbf{G}$. If $\mathbf{G} = \square$ then, by (2.5.5), $(pu(\mathsf{D}) \triangleright pu(\mathsf{D}'))(\square) = \{\square\} = (pu(\mathsf{D} \triangleright pu(\mathsf{D}')))(\square)$.

Otherwise let $\mathbf{G} := (\mathsf{A}, \mathbf{G}')$ and $\mathtt{d} \in (pu(\mathsf{D}) \triangleright pu(\mathsf{D}'))(\mathbf{G})$. Two cases arise.

$\mathtt{d} \in pu(\mathsf{D})(\mathbf{G})$ In this case, since $\triangleright$ is extensive and $\cdot$ and $\times$ are monotonic, $pu(\mathsf{D}) \sqsubseteq pu(\mathsf{D} \triangleright pu(\mathsf{D}'))$ and then $\mathtt{d} \in pu(\mathsf{D} \triangleright pu(\mathsf{D}'))(\mathbf{G})$.

$\mathtt{d} \notin pu(\mathsf{D})(\mathbf{G})$ Since $pu(\mathsf{D}')$ is closed under renaming,

$$\mathtt{d} = \mathtt{d}_1 :: \mathtt{d}_2, \tag{1}$$

where $\mathtt{d}_1 \in pu(\mathsf{D})(\mathbf{G})$, $\mathit{last}(\mathtt{d}_1) = \mathbf{B} \neq \square$ and $\mathtt{d}_2 \in pu(\mathsf{D}')(\mathbf{B})$. By (2.5.5) and since pu is closed under renaming, $\mathtt{d}_1 = \mathtt{d}_3 \wedge \mathtt{d}_4$, where $\mathtt{d}_3 \in (\mathsf{A} \cdot \mathsf{D})(\mathsf{A})$ and $\mathtt{d}_4 \in pu(\mathsf{D})(\mathbf{G}')$. Then, by definition of $\times$ and by Point 2 of Lemma 2.7.2, $\mathtt{d}_1 :: \mathtt{d}_2 \in (((\mathsf{A} \cdot \mathsf{D}) \times pu(\mathsf{D})) \triangleright pu(\mathsf{D}'))(\mathbf{G}) \subseteq (((\mathsf{A} \cdot \mathsf{D}) \triangleright pu(\mathsf{D}')) \times (pu(\mathsf{D}) \triangleright pu(\mathsf{D}')))(\mathbf{G})$. Therefore, by (1), by definition of $\times$ and since $\mathbf{G} = (\mathsf{A}, \mathbf{G}')$, there exist two renamed versions $\mathtt{d}_5$ and $\mathtt{d}_6$ of elements in $((\mathsf{A} \cdot \mathsf{D}) \triangleright pu(\mathsf{D}'))(\mathsf{A})$ and in $(pu(\mathsf{D}) \triangleright pu(\mathsf{D}'))(\mathbf{G}')$ respectively, such that

$$\mathtt{d} = \mathtt{d}_5 \wedge \mathtt{d}_6. \tag{2}$$

By inductive hypothesis, $\mathtt{d}_6$ is a renamed version of an element in $pu(\mathsf{D} \triangleright pu(\mathsf{D}'))(\mathbf{G}')$. Moreover (since $\mathtt{d}_5$ is a renamed version of an element in $((\mathsf{A} \cdot \mathsf{D}) \triangleright pu(\mathsf{D}'))(\mathsf{A})$), by Point 1 of Lemma 2.7.2, $\mathtt{d}_5$ is a renamed version of an element in $(\mathsf{A} \cdot (\mathsf{D} \triangleright pu(\mathsf{D}')))(\mathsf{A})$. Finally, by definition of $\times$ and pu and by (2), $\mathtt{d} \in ((\mathsf{A} \cdot (\mathsf{D} \triangleright pu(\mathsf{D}'))) \times pu(\mathsf{D} \triangleright pu(\mathsf{D}')))(\mathbf{G}) \subseteq pu(\mathsf{D} \triangleright pu(\mathsf{D}'))(\mathbf{G})$.

Point 2 The proof is by induction on $\mathtt{h}$. The case $\mathtt{h} = 0$ is straightforward, since $pu^0(\mathsf{D}) = \phi = pu_0(\mathsf{D})$.

Otherwise the proof is by structural induction on $\mathbf{G}$.

$\mathbf{G} = \square$ By (2.5.7) and (2.5.6), $pu^{\mathtt{h}}(\mathsf{D})(\square) = \{\square\} = pu_1(\mathsf{D})(\square)$.

$\underline{\mathbf{G} = \mathbf{A}, \mathbf{G}'}\rfloor$ Let $\mathtt{d} \in pu^{h}(\mathsf{D})(\mathbf{G}) = (pu(\mathsf{D} \triangleright pu^{h-1}(\mathsf{D})))(\mathbf{G})$. Then, by (2.5.7),

$$\mathtt{d} = \mathtt{d}_1 \wedge \mathtt{d}_2, \text{ where } \mathtt{d}_1 \in (\mathsf{A} \cdot (\mathsf{D} \triangleright pu^{h-1}(\mathsf{D})))(\mathsf{A}) \tag{3}$$

and $\mathtt{d}_2 \in pu^{h}(\mathsf{D})(\mathbf{G}')$. By inductive hypothesis, $\mathtt{d}_2 \in \sum\{pu_k(\mathsf{D})\}_{k\geq 0}(\mathbf{G}')$ and therefore there exists $m \geq 0$ such that

$$\mathtt{d}_2 \in pu_m(\mathsf{D})(\mathbf{G}'). \tag{4}$$

Moreover,

$$\begin{array}{ll}
\mathsf{A} \cdot (\mathsf{D} \triangleright pu^{h-1}(\mathsf{D})) = & [\text{by Point 1 of Lemma 2.7.2}] \\
(\mathsf{A} \cdot \mathsf{D}) \triangleright pu^{h-1}(\mathsf{D}) \sqsubseteq & [\text{by inductive hypothesis}] \\
(\mathsf{A} \cdot \mathsf{D}) \triangleright \sum\{pu_k(\mathsf{D})\}_{k\geq 0} \sqsubseteq & [\text{by (2.5.5)}] \\
pu(\mathsf{D}) \triangleright \sum\{pu_k(\mathsf{D})\}_{k\geq 0} = & [\text{by Lemma 2.2.7}] \\
\sum\{pu(\mathsf{D}) \triangleright pu_k(\mathsf{D})\}_{k\geq 0} = & [\text{by (2.5.6)}] \\
\sum\{pu_k(\mathsf{D})\}_{k\geq 0}. &
\end{array}$$

Then, by (3), there exists $l \geq 0$ such that

$$\mathtt{d}_1 \in pu_l(\mathsf{D})(\mathsf{A}). \tag{5}$$

Now, by definition of $\wedge$, we have two possibilities.

$\underline{\boldsymbol{last}(\mathtt{d}_1) \neq \square}\rfloor$ In this case $\mathtt{d} = \mathtt{d}_1 \wedge \mathbf{G}'$. Since $\mathtt{d}_2 \in pu_m(\mathsf{D})(\mathbf{G}')$, for any predicate symbol p occurring in $\mathbf{G}'$, $\mathsf{D}(\mathsf{p}(\boldsymbol{x})) \neq \emptyset$. Then $\mathbf{G}' \in pu_l(\mathsf{D})(\mathbf{G}')$ and therefore, by (2.5.5), by (5) and by definition of $\triangleright$, $\mathtt{d} = \mathtt{d}_1 \wedge \mathbf{G}' \in pu_l(\mathsf{D})(\mathbf{G}) \subseteq (\sum\{pu_k(\mathsf{D})\}_{k\geq 0})(\mathbf{G})$.

$\underline{\boldsymbol{last}(\mathtt{d}_1) = \square}\rfloor$ Let $\mathtt{d}_1 := \mathsf{A} \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \square$ and $\vartheta := \vartheta_1 \cdots \vartheta_k$. By definition of $\wedge$ and by (3),

$$\mathtt{d} = (\mathtt{d}_1 \wedge \mathbf{G}') :: \partial_\vartheta(\mathtt{d}_2). \tag{6}$$

Analogously to the previous case, using (5),

$$\mathtt{d}_1 \wedge \mathbf{G}' \in pu_l(\mathsf{D})(\mathbf{G}). \tag{7}$$

Moreover, by (6), $\partial_\vartheta(\mathtt{d}_2)$ is defined. Then, by (4), by properties of pu and by a straightforward inductive argument, $\partial_\vartheta(\mathtt{d}_2) \in pu_m(\mathsf{D})(\mathbf{G}'\vartheta)$. Hence, by definition of $\triangleright$, by (6) and (7), $\mathtt{d} = (\mathtt{d}_1 \wedge \mathbf{G}') :: \partial_\vartheta(\mathtt{d}_2) \in (pu_l(\mathsf{D}) \triangleright pu_m(\mathsf{D}))(\mathbf{G})$ and therefore, by Lemma 2.2.6 and by a straightforward inductive argument, $\mathtt{d} \in pu_{l+m}(\mathsf{D})(\mathbf{G})$. ■

Proof of Lemma 2.5.12. We prove the points separately.

Point 1 By definition of $\sqsubseteq$ it is sufficient to prove that, for any $\mathbf{G} \in \mathit{Goals}$, $su(\mathsf{D})(\mathbf{G}) \subseteq pu(\mathsf{D} + \mathit{Id}_{\mathbb{I}})(\mathbf{G})$. We distinguish two cases. If $\mathbf{G} = \square$, $su(\mathsf{D})(\square)$ is not defined and then the thesis follows trivially.

Otherwise let $\mathbf{G} = (\mathsf{A}, \mathbf{G}')$ and observe that, by (2.5.5), $\mathit{Id}_{\mathbb{C}} \sqsubseteq pu(\mathsf{D} + \mathit{Id}_{\mathbb{I}})$. Then the following facts hold

$$
\begin{array}{ll}
su(\mathsf{D})(\mathbf{G}) = & [\text{by (2.3.2)}] \\
((\mathsf{A} \cdot \mathsf{D}) \times \mathit{Id}_{\mathbb{C}})(\mathbf{G}) \subseteq & [\text{by the previous observation}] \\
((\mathsf{A} \cdot \mathsf{D}) \times pu(\mathsf{D} + \mathit{Id}_{\mathbb{I}}))(\mathbf{G}) \subseteq & [\text{since } \times \text{ is monotonic}] \\
((\mathsf{A} \cdot (\mathsf{D} + \mathit{Id}_{\mathbb{I}})) \times pu(\mathsf{D} + \mathit{Id}_{\mathbb{I}}))(\mathbf{G}) = & [\text{by definition of } pu] \\
pu(\mathsf{D} + \mathit{Id}_{\mathbb{I}})(\mathbf{G}). &
\end{array}
$$

Point 2 We prove (by induction on $\mathfrak{n}$) that, for any $\mathbf{G} = \mathsf{A}_1, \ldots, \mathsf{A}_{\mathfrak{n}} \in \mathit{Goals}$ and $\mathfrak{n} \geq 0$, $\mathcal{G}[\![\mathbf{G}]\!]_{\mathsf{D}} \sqsubseteq \mathit{Id}_{\mathbb{C}} + su_{\mathfrak{n}}(\mathsf{D})$. Then the thesis follows by (2.5.5).

If $\mathfrak{n} = 0$ the thesis follows trivially, since $\mathbf{G} = \square$ and, by definition, $\mathcal{G}[\![\square]\!]_{\mathsf{D}} = \phi_{\square} \sqsubseteq \mathit{Id}_{\mathbb{C}}$.

Otherwise let $\mathbf{G} := \mathsf{A}, \mathbf{G}'$. Then

$$
\begin{array}{ll}
\mathcal{G}[\![\mathsf{A}, \mathbf{G}']\!]_{\mathsf{D}} = & [\text{by definition of } \mathcal{G}[\![\cdot]\!] \text{ and } \mathcal{A}[\![\cdot]\!]] \\
(\mathsf{A} \cdot \mathsf{D}) \times \mathcal{G}[\![\mathbf{G}']\!]_{\mathsf{D}} \sqsubseteq & [\text{by inductive hypothesis}] \\
(\mathsf{A} \cdot \mathsf{D}) \times (\mathit{Id}_{\mathbb{C}} + su_{\mathfrak{n}-1}(\mathsf{D})). &
\end{array}
$$

Now let $\mathtt{d} \in \mathcal{G}[\![\mathbf{G}]\!]_{\mathsf{D}}(\mathbf{G})$. By the previous result and by definition of $\times$, $\mathtt{d} = \mathtt{d}_1 \wedge \mathtt{d}_2$, where $\mathtt{d}_1 \in (\mathsf{A} \cdot \mathsf{D})(\mathsf{A})$ and $\mathtt{d}_2 \in (\mathit{Id}_{\mathbb{C}} + su_{\mathfrak{n}-1}(\mathsf{D}))(\mathbf{G}')$. Two cases arise.

last$(\mathtt{d}_1) \neq \square$ In this case, by definition of $\wedge$, $\mathtt{d} = \mathtt{d}_1 \wedge \mathbf{G}' \in ((\mathsf{A} \cdot \mathsf{D}) \times \mathit{Id}_{\mathbb{C}})(\mathbf{G})$ and therefore, by (2.3.2), $\mathtt{d} \in su(\mathsf{D})(\mathbf{G})$. Then, by definition of $\triangleright$ and of $+$, $\mathtt{d} \in (\mathit{Id}_{\mathbb{C}} + su_{\mathfrak{n}}(\mathsf{D}))(\mathbf{G})$.

last$(\mathtt{d}_1) = \square$ Let $\mathtt{d}_1 := \mathsf{A} \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_k]{\vartheta_k} \square$ and $\vartheta := \vartheta_1 \cdots \vartheta_k$. By definition of $\wedge$, $\mathtt{d} = (\mathtt{d}_1 \wedge \mathbf{G}') :: \partial_{\vartheta}(\mathtt{d}_2)$. Then, since $\mathtt{d}_1 \in (\mathsf{A} \cdot \mathsf{D})(\mathsf{A})$, $\mathtt{d}_1 \wedge \mathbf{G}' \in ((\mathsf{A} \cdot \mathsf{D}) \times \mathit{Id}_{\mathbb{C}})(\mathbf{G})$ and therefore, by (2.3.2), $\mathtt{d}_1 \wedge \mathbf{G}' \in su(\mathsf{D})(\mathbf{G})$. Moreover, since $\partial_{\vartheta}(\mathtt{d}_2)$ is defined and $\mathtt{d}_2 \in (\mathit{Id}_{\mathbb{C}} + su_{\mathfrak{n}-1}(\mathsf{D}))(\mathbf{G}')$, by properties of su and by a straightforward inductive argument, $\partial_{\vartheta}(\mathtt{d}_2) \in (\mathit{Id}_{\mathbb{C}} + su_{\mathfrak{n}-1}(\mathsf{D}))(\mathbf{G}'\vartheta)$. By the previous results and by definition of $\triangleright$,

$$
\mathtt{d} = (\mathtt{d}_1 \wedge \mathbf{G}') :: \partial_{\vartheta}(\mathtt{d}_2) \in (su(\mathsf{D}) \triangleright (\mathit{Id}_{\mathbb{C}} + su_{\mathfrak{n}-1}(\mathsf{D})))(\mathbf{G}). \tag{1}
$$

Finally observe that, since $\mathsf{n} \geq 1$, $su(\mathsf{D}) \sqsubseteq su_{\mathsf{n}}(\mathsf{D})$. Then

$$\begin{aligned}
&su(\mathsf{D}) \triangleright (Id_{\mathbb{C}} + su_{\mathsf{n}-1}(\mathsf{D})) = \\
&\qquad [\text{by Lemma 2.2.7}] \\
&(su(\mathsf{D}) \triangleright Id_{\mathbb{C}}) + (su(\mathsf{D}) \triangleright su_{\mathsf{n}-1}(\mathsf{D})) = \\
&\qquad [\text{since } \mathsf{D} \triangleright Id_{\mathbb{C}} = \mathsf{D}] \\
&su(\mathsf{D}) + (su(\mathsf{D}) \triangleright su_{\mathsf{n}-1}(\mathsf{D})) = \\
&\qquad [\text{by (2.2.7) and the previous observation}] \\
&su_{\mathsf{n}}(\mathsf{D}).
\end{aligned}$$

Therefore, by (1), $\mathsf{d} \in su_{\mathsf{n}}(\mathsf{D})(\mathbf{G}) \subseteq (Id_{\mathbb{C}} + su_{\mathsf{n}}(\mathsf{D}))(\mathbf{G})$. ■

Proof of Corollary 2.5.13. The inclusion $\sqsupseteq$ is straightforward by Corollary 2.5.11 and by Point 1 of Lemma 2.5.12. For the other inclusion we prove that, for any $\mathsf{h} \geq 0$, $pu_{\mathsf{h}}(Id_{\mathbb{I}} + \mathsf{D}) \sqsubseteq Id_{\mathbb{C}} + \sum\{su_{\mathsf{k}}(\mathsf{D})\}_{\mathsf{k}\geq 0}$. Then the thesis follows by Corollary 2.5.11. For $\mathsf{h} = 0$ the proof is trivial, since $pu_0(Id_{\mathbb{I}} + \mathsf{D}) = \phi$.

Otherwise we prove the statement by induction on $\mathsf{h} \geq 1$.

$\underline{\mathsf{h} = 1}$ Straightforward by Point 2 of Lemma 2.5.12.

$\underline{\mathsf{h} > 1}$ Let $\mathbf{G}$ be a goal and $\mathsf{d} \in pu_{\mathsf{h}}(Id_{\mathbb{I}} + \mathsf{D})(\mathbf{G})$. By (2.5.6) and by Point 2 of Lemma 2.2.6, $\mathsf{d} \in (pu_{\mathsf{h}-1}(Id_{\mathbb{I}} + \mathsf{D}) \triangleright pu(Id_{\mathbb{I}} + \mathsf{D}))(\mathbf{G})$. If $\mathsf{d} \in pu_{\mathsf{h}-1}(Id_{\mathbb{I}} + \mathsf{D})(\mathbf{G})$ then the thesis follows by inductive hypothesis (and definition of $\sqsubseteq$).

Otherwise, by definition of $\triangleright$, we can assume that

$$\begin{aligned}
&\mathsf{d} = \mathsf{d}_1 :: \mathsf{d}_2, \text{ where } \mathsf{d}_1 \in pu_{\mathsf{h}-1}(Id_{\mathbb{I}} + \mathsf{D})(\mathbf{G}), \\
&last(\mathsf{d}_1) = \mathbf{B} \neq \Box \text{ and } \mathsf{d}_2 \in pu(Id_{\mathbb{I}} + \mathsf{D})(\mathbf{B}).
\end{aligned} \tag{1}$$

By inductive hypothesis, $\mathsf{d}_1 \in (Id_{\mathbb{C}} + \sum\{su_{\mathsf{k}}(\mathsf{D})\}_{\mathsf{k}\geq 0})(\mathbf{G})$ and then there exists $\mathsf{n} \geq 0$ such that

$$\mathsf{d}_1 \in (Id_{\mathbb{C}} + su_{\mathsf{n}}(\mathsf{D}))(\mathbf{G}). \tag{2}$$

Moreover, by Point 2 of Lemma 2.5.12 and by (1), $\mathsf{d}_2 \in (Id_{\mathbb{C}} + (\sum\{su_{\mathsf{k}}(Id_{\mathbb{I}} + \mathsf{D})\}_{\mathsf{k}\geq 0})(\mathbf{B})$ and therefore there exists $\mathsf{m} \geq 0$ such that

$$\mathsf{d}_2 \in (Id_{\mathbb{C}} + (su_{\mathsf{m}}(Id_{\mathbb{I}} + \mathsf{D})))(\mathbf{B}). \tag{3}$$

Now observe that, by Lemma 2.2.7 and since $Id_{\mathbb{C}} = su(Id_{\mathbb{I}}) + \phi_{\Box}$, $su(Id_{\mathbb{I}} + \mathsf{D}) = su(Id_{\mathbb{I}}) + su(\mathsf{D}) = Id_{\mathbb{C}} + su(\mathsf{D})$. Then (since $\forall \mathsf{D} \in \mathbb{C}.\, \mathsf{D} \triangleright Id_{\mathbb{C}} = \mathsf{D}$, $Id_{\mathbb{C}} \triangleright \mathsf{D} = Id_{\mathbb{C}} + \mathsf{D}$) $Id_{\mathbb{C}} + su_{\mathsf{m}}(Id_{\mathbb{I}} + \mathsf{D}) = Id_{\mathbb{C}} + \sum\{su_{\mathsf{k}}(\mathsf{D})\}_{\mathsf{k}\leq \mathsf{m}} + \sum\{Id_{\mathbb{C}} +$

$su_{\mathsf{k}}(\mathsf{D})\}_{\mathsf{k}<\mathsf{m}}$. Hence (since $\forall \mathsf{D} \in \mathbb{C}, \mathsf{k} \leq \mathsf{k}'.\ su_{\mathsf{k}}(\mathsf{D}) \sqsubseteq su_{\mathsf{k}'}(\mathsf{D}))\ Id_{\mathbb{C}} + su_{\mathsf{m}}(Id_{\mathbb{I}} + \mathsf{D}) = Id_{\mathbb{C}} + su_{\mathsf{m}}(\mathsf{D})$ and therefore, by (3), $\mathtt{d}_2 \in (Id_{\mathbb{C}} + su_{\mathsf{m}}(\mathsf{D}))(\mathbf{B})$. Then the following holds.

$\mathtt{d} \in$

[by (1), by definition of $\triangleright$, by (2) and last result]

$((Id_{\mathbb{C}} + su_{\mathsf{n}}(\mathsf{D})) \triangleright (Id_{\mathbb{C}} + su_{\mathsf{m}}(\mathsf{D})))(\mathbf{G}) =$

[by Lemma 2.2.7 and since $\triangleright$ is extensive]

$(Id_{\mathbb{C}} + (su_{\mathsf{n}}(\mathsf{D}) \triangleright Id_{\mathbb{C}}) + (su_{\mathsf{n}}(\mathsf{D}) \triangleright su_{\mathsf{m}}(\mathsf{D})))(\mathbf{G}) =$

[since $\forall \mathsf{D} \in \mathbb{C}.\ \mathsf{D} \triangleright Id_{\mathbb{C}} = \mathsf{D}$ and since $\triangleright$ is extensive]

$(Id_{\mathbb{C}} + su_{\mathsf{n}}(\mathsf{D}) \triangleright su_{\mathsf{m}}(\mathsf{D}))(\mathbf{G}) =$

[by (2.2.7)]

$(Id_{\mathbb{C}} + su_{\mathsf{n}+\mathsf{m}}(\mathsf{D}))(\mathbf{G}) \subseteq$

[by definition of $\sum$]

$(Id_{\mathbb{C}} + \sum\{su_{\mathsf{k}}(\mathsf{D})\}_{\mathsf{k}\geq 0})(\mathbf{G})$.

■

Chapter 3

The Abstraction Framework

Definite logic programs have a very elegant declarative semantics, i.e., the least Herbrand model. However, some semantics-based techniques (such as program analysis, debugging and transformation) require more traditional semantics which are able to capture computational rather than declarative properties. Semantic definitions can be different in style, as in the case of the top-down *SLD*-resolution *operational semantics* and the bottom-up fixpoint *denotational semantics*. They can be different because of some of their properties. For example, *SLD*-resolution is *goal-dependent* since it allows us to compute a denotation for a given goal. The fixpoint semantics is instead *goal-independent*, since it provides a denotation for a set of procedure declarations.

Some important properties of a semantics can be described as *compositionality* properties. One example is *OR-compositionality*, which tells us that the denotation of a set of clauses can be obtained by composing the denotations of the clauses. Most of the existing goal-independent semantics, such as the standard fixpoint semantics, are not *OR*-compositional. However the most relevant difference is related to the *observable* the semantics is intended to model. An observable is any property which can be observed in an *SLD*-tree. Some observables model declarative properties. An example is correct answer substitutions. However most useful observables model operational properties. Examples are resultants, proof trees, finite failures, computed answer substitutions, partial answers, call patterns, types and groundness dependencies.

Several ad-hoc semantics modeling various observables have been defined. These include correct answer substitutions [16, 48], computed answer substitutions [45], partial answers [43], *OR*-compositional correct answers [56], *OR*-compositional computed answers [14], call patterns [55], proof trees [66, 67] and resultants [54]. In addition there are several semantics specifically designed for static program analysis, which can handle various observables such as types and groundness dependencies.

A framework where one can define denotations modeling various observables (thus inheriting the basic constructions and results) was given in [53], by defining the observables by means of equivalence relations. More general semantic frameworks, which can also take into account approximation, can be defined by using *abstract interpretation* [32, 34], a theory which was developed to reason about the relation among different semantics, including the approximate semantics useful for static program analysis. This is the approach taken in [20], where an observable is an abstraction according to abstract interpretation theory, and in [58], where abstract interpretation is used to discuss the relation among different semantics.

In this thesis we push forward the approach in [20], by defining a semantic framework whose ingredients are, as in the case of most abstract interpretation frameworks, a concrete semantics and an observable. Our concrete semantics (Chapter 2) models *SLD*-trees and is formalized both denotationally and operationally. It's main properties (see Section 2.5) are

- equivalence between operational semantics and denotational semantics,
- existence of a goal-independent denotation for a set of definite clauses, defined in terms of a transition system, equivalent to the (denotational) fixpoint semantics,
- correctness and minimality (w.r.t. *SLD*-trees), *AND*-compositionality and *OR*-compositionality of the goal-independent denotations.

An observable (Section 3.1) is a Galois insertion between the domain of *SLD*-trees and an abstract domain describing the properties to be modeled. The abstract denotational definition, transition system and goal-independent denotations are systematically derived from the concrete ones, by replacing the concrete semantic operators by their optimal abstract versions (Section 3.2). Therefore, the definition style (denotational semantics and transition system) of the concrete semantics will be inherited by all the abstract semantics.

The next step is the definition of a taxonomy of classes of observables. An observable belongs to a class if it satisfies a set of conditions relating the concrete semantic operators and the Galois insertion. Once we have shown that an observable belongs to a given class, we know how to automatically derive the "best" semantics and which are the properties of such a semantics. The properties we consider include precision, relation between abstract operational semantics and abstract denotational semantics, existence of a goal-independent denotation for a set of definite clauses, correctness, minimality and compositionality w.r.t. various syntactic operators.

The first class we consider is the one of *perfect observables* (Section 3.3). We prove that perfect observables are precise and have all the properties of the concrete semantics. We show that this class includes resultants and proof trees.

For the class of *denotational observables* (Section 3.4), we can obtain the optimal abstract semantics only in a denotational way, by taking the optimal abstract version of the semantic operator defining the denotation of the clauses (roughly speaking, the immediate consequence operator). The abstract operational semantics is less precise. We prove that denotational observables have a precise abstract denotational semantics and that the abstract (goal-independent) denotation is correct, minimal and *AND*-compositional. Therefore, by moving from perfect to denotational observables, we lose the precision of the abstract transition system and *OR*-compositionality. We show that the class includes computed answer substitutions and call patterns.

The third class of observables we study is the class of *semi-denotational observables* (Section 3.5), intended to model some of the properties useful for static program analysis, where we give up precision to achieve termination in the construction of the abstract semantics. The semantics construction of semi-denotational observables is the same of denotational observables. We just lose the precision of the abstract denotational semantics (which is in any case more precise than the operational one). We formally show that the class includes the domain $\mathcal{POS}$ for groundness analysis and the domain $depth(\mathsf{k})$.

Finally, we consider the class of semi-perfect observables (Section 3.6) which allow us to handle approximate properties in an operational way and to model top-down program analysis. These observables have all the properties of perfect observables apart from precision. In particular, they have equivalent operational and denotational semantics, and the (top-down and bottom-up) goal-independent denotations are *AND*-compositional and *OR*-compositional. Let us just note that semi-perfect observables are essentially the observables which model top-down abstract interpretation frameworks (for example, [15, 69]).

In Section 3.7 we show how our results give some new insights into some classical controversial issues, such as top-down analysis versus bottom-up analysis and *goal-dependence* versus *goal-independence*. Finally, in Section 3.8, we discuss some practical applications (in particular to program diagnosis, which is discussed extensively in Chapter 5) and possible extensions.

The results of this chapter are from [22, 21].

3.1 The Observables

The properties found in Chapter 2 for $\mathcal{O}[\![\mathsf{P}]\!]$ and $\mathcal{F}[\![\mathsf{P}]\!]$ allow us to claim that we have a good denotation modeling *SLD*-trees. Our goal however is to find the same results for the denotations modeling more abstract observables. We want then to develop a

theory according to which the semantic properties of *SLD*-trees shown in Chapter 2 are inherited by the denotations which model abstractions of the *SLD*-trees. We will model the abstractions by using the Abstract Interpretation theory [34].

An observable property domain is a set of properties of derivations with an ordering relation which can be viewed as an approximation structure. An observation consists of looking at an *SLD*-tree, and then extracting some property (abstraction). Since any *SLD*-tree is (isomorphic to) a collection, an observable is a function from $\mathbb{C}$ to a suitable property domain $\mathcal{D}$, which preserves the approximation structure. Such a function must be a Galois insertion.

Let $(\mathcal{D}, \preceq)$ be a complete lattice. A function $\alpha : \mathit{WFS} \to \mathcal{D}$ is a *domain abstraction* if there exists γ such that $\langle \alpha, \gamma \rangle : (\mathit{WFS}, \subseteq) \rightleftharpoons (\mathcal{D}, \preceq)$ is a Galois insertion. Given an abstract domain $\mathcal{D}$ we are generally interested in the *abstract behavior* of the queries, which are elements of a domain of partial functions $\mathbb{A} \subseteq [\mathit{Goals} \rightharpoonup \mathcal{D}]$ (ordered by the trivial extension $\leq$ of $\preceq$) and are called $\mathbb{A}$-*collections*.

The insertion $\langle \alpha, \gamma \rangle$ can be *lifted* to collections by defining[1] $\forall \mathsf{D} \in \mathbb{C}.\, \alpha^\star(\mathsf{D}) := \lambda \mathbf{G}.\, \alpha(\mathsf{D}(\mathbf{G}))$, $\mathbb{A} := \alpha^\star(\mathbb{C})$ and $\forall \mathsf{S} \in \mathbb{A}.\, \gamma^\star(\mathsf{S}) := \lambda \mathbf{G}.\ \mathit{wf}_{\mathbf{G}}(\gamma(\mathsf{S}(\mathbf{G})))$, where $\mathit{wf}_{\mathbf{G}}(\mathsf{S})$ is the greatest well-formed subset of any set of derivations S, restricted to the derivations starting from $\mathbf{G}$ only. The pair $\langle \alpha^\star, \gamma^\star \rangle : (\mathbb{C}, \sqsubseteq) \rightleftharpoons (\mathbb{A}, \leq)$ is a Galois insertion. From now on we will often abuse notation and denote $\alpha^\star$ by α. Furthermore, if there exists a bijective Galois insertion between two domains, we identify them. As in the concrete case, a *pure* $\mathbb{A}$-collection is any element of $\mathsf{X} \in \mathbb{A}$ which is defined for pure atomic goals only. We denote by $\mathbb{PA}$ the sub-lattice of pure $\mathbb{A}$-collections.

Definition 3.1.1 *Let $(\mathbb{A}, \leq)$ be a complete lattice of $\mathbb{A}$-collections. A function $\alpha : \mathbb{C} \to \mathbb{A}$ is an* observable *if it maps finite elements of $\mathbb{C}$ to finite elements*[2] *of $\mathbb{A}$ and there exists γ such that*

1. $\langle \alpha, \gamma \rangle : (\mathbb{C}, \sqsubseteq) \rightleftharpoons (\mathbb{A}, \leq)$ *is a Galois insertion,*
2. $\alpha(\mathbb{PC}) = \mathbb{PA}$ *and* $\gamma(\mathbb{PA}) \subseteq \mathbb{PC}$,
3. $\forall \mathsf{D}, \mathsf{D}' \in \mathbb{PC}.\, \mathsf{D} \equiv_{\mathbb{C}} \mathsf{D}' \Longrightarrow (\gamma\alpha)(\mathsf{D}) \equiv_{\mathbb{C}} (\gamma\alpha)(\mathsf{D}')$.

Note that given a domain abstraction it is easy to obtain an observable by the above mentioned lifting[3].

[1] Remember that if $\mathsf{D}(\mathbf{G})$ is undefined then also $\alpha(\mathsf{D}(\mathbf{G}))$ is undefined.

[2] Let $\mathbb{A} \subseteq [\mathit{Goals} \rightharpoonup \mathcal{D}]$. We assume that the elements of the domain $\mathcal{D}$ can be represented by means of a syntactic expression built over the (free) variables which appear in the corresponding derivations in *WFS*. By *finite element* of $\mathcal{D}$ we mean any element which is finitely representable in the domain $\mathcal{D}$. Indeed the only relevant thing to our purposes is that a finite element has a finite number of free variables.

[3] All the observables defined in the examples are indeed obtained by lifting a domain abstraction.

We can define an *abstract enhanced variance relation* $\equiv_{\mathbb{A}}$ on $\mathbb{A}$-collections as follows: for any $\mathbb{A}$-collections X, X', $\mathsf{X} \equiv_{\mathbb{A}} \mathsf{X}' \Longleftrightarrow \gamma(\mathsf{X}) \equiv_{\mathbb{C}} \gamma(\mathsf{X}')$.

Definition 3.1.2 *An* $\mathbb{A}$-interpretation *is a pure* $\mathbb{A}$*-collection modulo* $\equiv_{\mathbb{A}}$*. We denote by* $(\mathbb{I}_{\mathbb{A}}, \leq)$ *the complete lattice of* $\mathbb{A}$*-interpretations with the induced quotient order.*

Point 3 of Definition 3.1.1 states that the observation does not depend on the choice of the variable names and on the choice of the *mgus* used in the derivations. Namely $\mathsf{D} \equiv_{\mathbb{C}} \mathsf{D}'$ implies $\alpha(\mathsf{D}) \equiv_{\mathbb{A}} \alpha(\mathsf{D}')$. Hence, for any $\mathbb{C}$-interpretation I, the $\mathbb{A}$-interpretation $\alpha(\mathrm{I})$ is well defined[4].

Each observable α induces an *observational equivalence* $\approx_\alpha$ on programs. Namely $\mathsf{P}_1 \approx_\alpha \mathsf{P}_2$ if and only if, for all $\mathbf{G} \in \mathit{Goals}$,

$$\alpha\left(\mathcal{B}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}_1]\!]\right) = \alpha\left(\mathcal{B}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}_2]\!]\right), \tag{3.1.1}$$

i.e., if P_1 and P_2 cannot be distinguished by looking at the abstraction of their concrete behaviors. Note that the abstract behavior of a query, as defined in Section 3.2, will in general be less precise than the abstraction of the concrete behavior $\alpha(\mathcal{B}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}]\!])$, which is therefore sometimes referred to as the *most precise abstract behavior*.

Note that whenever we have two equivalent observables, the induced equivalences on programs are the same, but the converse does not hold. For example $\mathsf{P}_1 \approx \mathsf{P}_2$ if and only if $\mathsf{P}_1 \equiv \mathsf{P}_2$. Therefore, the equivalence on programs induced by the *SLD*-tree semantics is the same as the syntactical equivalence on programs (variance). However, we can syntactically observe many more properties in *SLD*-trees than in programs themselves.

Example 3.1.3 (Computed Answer Substitutions)

If ξ denotes *computed answer substitution abstraction* we must consider the domain $(\wp(\mathit{Subst}), \subseteq)$ and define the domain abstraction $\xi : \mathit{WFS} \to \wp(\mathit{Subst})$ as $\xi(\mathsf{S}) := \{\mathit{answer}(\mathtt{d}) \mid \mathtt{d} \in \mathsf{S}, \mathit{last}(\mathtt{d}) = \square\}$. This abstraction can be lifted to the abstract domain $\mathbb{A}_{\mathrm{ca}} \subseteq [\mathit{Goals} \rightharpoonup \wp(\mathit{Subst})]$ by defining $\langle \xi, \xi^\gamma \rangle : \mathbb{C} \rightleftharpoons \mathbb{A}_{\mathrm{ca}}$ as

$$\begin{aligned}
\xi(\mathsf{D}) &:= \lambda\mathbf{G}.\left\{\mathit{answer}(\mathtt{d}) \mid \mathtt{d} \in \mathsf{D}(\mathbf{G}),\ \mathit{last}(\mathtt{d}) = \square\right\}.\\
\xi^\gamma(\mathsf{X}) &:= \lambda\mathbf{G}.\left\{\mathtt{d} \mid \mathit{first}(\mathtt{d}) = \mathbf{G},\ \mathit{last}(\mathtt{d}) = \square,\ \mathit{answer}(\mathtt{d}) \in \mathsf{X}(\mathbf{G})\right\} \cup\\
&\qquad \left\{\mathtt{d} \mid \mathit{first}(\mathtt{d}) = \mathbf{G},\ \mathit{last}(\mathtt{d}) \neq \square\right\}.
\end{aligned}$$

$\xi : \mathbb{C} \to \mathbb{A}_{\mathrm{ca}}$ is an observable (the proof is in Section 3.9). Then we can define the abstract enhanced variance relation $\equiv_{\mathbb{A}_{\mathrm{ca}}}$ on $\mathbb{A}_{\mathrm{ca}}$ as mentioned before. By using the

[4] We can take the abstraction, by means of α, of any representative of I. By Point 3 of Definition 3.1.1 it is a representative of the intended $\mathbb{A}$-interpretation.

same arguments of the proof that ξ is an observable, it is easy to check that $X \equiv_{\mathbb{A}_{ca}} X'$ if and only if, for any $p(\mathbf{x})$, there exists $p(\mathbf{y})$ such that (if $X(p(\mathbf{x}))$ is defined then $X'(p(\mathbf{y}))$ is defined and), for any $\vartheta \in X(p(\mathbf{x}))$, there exists $\vartheta' \in X'(p(\mathbf{y}))$ such that $p(\mathbf{x})\vartheta \equiv p(\mathbf{y})\vartheta'$ and vice versa. It is easy to see that $P_1 \approx_\xi P_2$ if and only if for any goal $\mathbf{G}$, $\mathbf{G}$ has the same (modulo variance) computed answer substitutions in P_1 and in P_2.

We show now the result of application of the ξ observable to some of the collections introduced in examples of Chapter 2.

Let D_1, D_2 be the collections of Example 2.1.1 (on page 39). Then

$$\xi(D_1) = \begin{cases} p(x,y) \mapsto \{\{x/g(a), y/b\}\} \\ n(x) \quad \mapsto \emptyset \end{cases}$$

$$\xi(D_2) = \begin{cases} q(x,y) \quad \mapsto \{\{x/a\}\} \\ q(f(x),z) \mapsto \{\{x/g(s), z/a\}\} \end{cases}$$

Consider now the collection $\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ P]\!]$ of Example 2.2.1 (on page 43), where P is the append program and $\mathbf{G} := ap([a],[l],x), ap(x,[h],z)$. Then

$$\xi(\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ P]\!]) = ap([a],[l],x), ap(x,[h],z) \mapsto \{\{x/[a,l], z/[a,l,h]\}\}$$

Let D'''' be the collection of Example 2.3.1 (on page 54). Its abstraction is

$$\xi(D'''') = anc(x,y) \mapsto \{\{x/jay, y/tim\}, \{x/joe, y/jay\}, \{x/joe, y/tim\}\}$$

while the abstraction of the collection $D_1 \triangleright D_2$ of Example 2.2.5 (on page 49) is

$$\xi(D_1 \triangleright D_2) = \begin{cases} p(x,y) \mapsto \{\{x/g(a), y/b\}, \{x/a\}\} \\ n(x) \quad \mapsto \emptyset \end{cases}$$

Note that, since $D_1 \sqsubseteq D_1 \triangleright D_2$, the theory requires $\xi(D_1) \leq \xi(D_1 \triangleright D_2)$, which is actually the case since $\xi(D_1)(n(x)) = \xi(D_1 \triangleright D_2)(n(x))$ and $\xi(D_1)(p(x,y)) \subset \xi(D_1 \triangleright D_2)(p(x,y))$.

3.2 From the Observables to the Abstract Semantics

Once we have an observable $\alpha : \mathbb{C} \to \mathbb{A}$, we want to derive systematically the abstract semantics. The idea is to define the optimal abstract versions of the various semantic operators and then check under which conditions (on the observable) we obtain the

optimal abstract semantics. This will allow us to identify some interesting classes of observables.

We will start by defining the optimal abstract counterparts of the basic operators defined on $\mathbb{C}$. Hence $\forall X, X', X_i \in \mathbb{A}$

$$A \tilde{\cdot} X := \alpha(A \cdot \gamma(X)), \tag{3.2.1}$$

$$X \tilde{\times} X' := \alpha(\gamma(X) \times \gamma(X')), \tag{3.2.2}$$

$$X \tilde{\triangleright} X' := \alpha(\gamma(X) \triangleright \gamma(X')), \tag{3.2.3}$$

$$\widetilde{\sum} \{X_i\}_{i \in I} := \alpha\left(\sum \{\gamma(X_i)\}_{i \in I}\right). \tag{3.2.4}$$

Once we have the optimal abstract operators, we can define the corresponding abstract semantics, obtained from the denotational and operational semantics of *SLD*-trees (of Chapter 2) by replacing the basic semantic operators by their optimal abstract versions.

Unfoldings

$$su_\alpha(X) := \widetilde{\sum} \left\{(A \tilde{\cdot} X) \tilde{\times} \alpha(Id_{\mathbb{C}})\right\}_{A \in Atoms} \tag{3.2.5}$$

$$pu_\alpha(X) := \widetilde{\sum} \left\{ \mathcal{G}_\alpha[\![\mathbf{G}]\!]_X \right\}_{\mathbf{G} \in Goals} \tag{3.2.6}$$

$$unf^k_{P,\alpha}(X) := \begin{cases} unf^{k-1}_{P,\alpha}(X) \tilde{\triangleright} su_\alpha(\alpha(tree(P))) & \text{if } k > 0 \\ X & \text{otherwise} \end{cases} \tag{3.2.7}$$

Denotational Semantics

$$\mathcal{Q}_\alpha[\![\mathbf{G}\ in\ P]\!] := \mathcal{G}_\alpha[\![\mathbf{G}]\!]_{lfp\, \mathcal{P}_\alpha[\![P]\!]} \tag{3.2.8}$$

$$\mathcal{G}_\alpha[\![A, \mathbf{G}]\!]_X := \mathcal{A}_\alpha[\![A]\!]_X \tilde{\times} \mathcal{G}_\alpha[\![\mathbf{G}]\!]_X \qquad \mathcal{G}_\alpha[\![\square]\!]_X := \alpha(\phi_\square) \tag{3.2.9}$$

$$\mathcal{A}_\alpha[\![A]\!]_X := A \tilde{\cdot} X \tag{3.2.10}$$

$$\mathcal{P}_\alpha[\![\{c\} \cup P]\!]_X := \mathcal{C}_\alpha[\![c]\!]_X \tilde{+} \mathcal{P}_\alpha[\![P]\!]_X \qquad \mathcal{P}_\alpha[\![\emptyset]\!]_X := \alpha(Id_{\mathbb{I}}) \tag{3.2.11}$$

$$\mathcal{C}_\alpha[\![H \leftarrow \mathbf{B}]\!]_X := \alpha(tree(H \leftarrow \mathbf{B})) \tilde{\triangleright} \mathcal{G}_\alpha[\![\mathbf{B}]\!]_X, \tag{3.2.12}$$

$$\mathcal{F}_\alpha[\![P]\!] := lfp\, \mathcal{P}_\alpha[\![P]\!]. \tag{3.2.13}$$

Operational Semantics

$$\frac{X \in \mathbb{A}, \quad X \neq X \tilde{\triangleright} su_\alpha(\alpha(tree(P)))}{X \xmapsto[P]{\alpha} X \tilde{\triangleright} su_\alpha(\alpha(tree(P)))} \tag{3.2.14}$$

$$\mathcal{B}_\alpha[\![\mathbf{G}\ in\ P]\!] := \widetilde{\sum} \left\{X \mid \alpha(\phi_{\mathbf{G}}) \xmapsto[P]{\alpha}{}^* X\right\} \tag{3.2.15}$$

$$\mathcal{O}_\alpha[\![P]\!] := \widetilde{\sum} \left\{ \mathcal{B}_\alpha[\![p(\mathbf{x})\ in\ P]\!]/_{\equiv_{\mathbb{A}}} \right\}_{p(\mathbf{x}) \in Goals}. \tag{3.2.16}$$

Any $\mathbb{A}$-interpretation X of $\mathbb{I}_{\mathbb{A}}$ is implicitly considered also as an arbitrary $\mathbb{A}$-collection obtained by choosing an arbitrary representative of X. By the following Lemma 3.2.1 and a straightforward structural induction, all the semantic operators that we have just introduced on $\mathbb{A}$-interpretations are independent from the choice of the representative. This is the reason why we defined the operators on $\mathbb{I}_{\mathbb{A}}$ in terms of their counterparts defined on $\mathbb{A}$, independently from the choice of the representative.

Lemma 3.2.1 *Let* $X, X' \in \mathbb{P}\mathbb{A}$. *If* $X \equiv_{\mathbb{A}} X'$ *then* $A \tilde{\cdot} X = A \tilde{\cdot} X'$.

Proof. $X \equiv_{\mathbb{A}} X'$ implies [by definition] $\gamma(X) \equiv_{\mathbb{C}} \gamma(X')$ and therefore [by (2.2.6)] $A \cdot \gamma(X) = A \cdot \gamma(X')$. Now [by applying α] we obtain $\alpha(A \cdot \gamma(X)) = \alpha(A \cdot \gamma(X'))$ which is the thesis. ■

Note that, by definition of $unf_{P,\alpha}$, $\xrightarrow[P]{\alpha}$ and $\mathcal{B}_\alpha[\![\cdot]\!]$,

$$\mathcal{B}_\alpha[\![\mathbf{G}\ in\ P]\!] = \widetilde{\sum} \left\{ unf^k_{P,\alpha}(\alpha(\phi_{\mathbf{G}})) \right\}_{k \geq 0} \tag{3.2.17}$$

$$\mathcal{O}_\alpha[\![P]\!] = \widetilde{\sum} \left\{ \left[\widetilde{\sum} \left\{ unf^k_{P,\alpha}(\alpha(\phi_{p(\mathbf{x})})) \right\}_{k \geq 0} \right] /_{\equiv_{\mathbb{A}}} \right\}_{p(\mathbf{x}) \in Goals}. \tag{3.2.18}$$

We are looking for conditions which guarantee that the abstract definitions of the denotations ((3.2.8)–(3.2.16)) do not lead to a loss of precision. Depending on these conditions we can characterize various classes of observables.

3.3 Perfect Observables

The first class of observables we consider is the one for which both the abstract denotational and the abstract operational semantics are precise. As a consequence, we can equivalently compute the abstract property in a top-down and in a bottom-up way, by mimicking the concrete computations. First of all recall that any observable is precise w.r.t. the sum operation, since for any Galois insertion,

$$\alpha(\sum \{D_i\}_{i \in I}) = \alpha(\sum \{(\gamma \circ \alpha)(D_i)\}_{i \in I}). \tag{3.3.1}$$

Definition 3.3.1 *Let* $\alpha : \mathbb{C} \to \mathbb{A}$ *be an observable. Then* α *is a* perfect observable *if*

$$\alpha(A \cdot D) = \alpha(A \cdot (\gamma \circ \alpha)D), \tag{3.3.2}$$

$$\alpha(D_1 \times D_2) = \alpha((\gamma \circ \alpha)D_1 \times (\gamma \circ \alpha)D_2), \tag{3.3.3}$$

$$\alpha(D_1 \triangleright D_2) = \alpha((\gamma \circ \alpha)D_1 \triangleright (\gamma \circ \alpha)D_2). \tag{3.3.4}$$

Example 3.3.2 (Computed Resultants)

Resultants are formulas of the form $\mathbf{G} \leftarrow \mathbf{B}$, which represent the relation between the initial goal and any intermediate goal in an *SLD*-derivation. Resultants have been introduced to prove the correctness of *SLD*-resolution [3]. A semantics modeling computed resultants was defined in [54]. Let *Res* be the set of resultants and consider the domain $\mathbb{A}_{\mathrm{cr}} \subseteq [\mathit{Goals} \rightharpoonup \mathit{Res}]$. The *computed resultants* observable $\chi : \mathbb{C} \to \mathbb{A}_{\mathrm{cr}}$ is defined by

$$\chi(\mathsf{D}) := \lambda\mathbf{G}.\big\{\mathbf{G}\vartheta \leftarrow \mathbf{B} \mid \mathtt{d} \in \mathsf{D}(\mathbf{G}),\ \mathbf{B} = \mathit{last}(\mathtt{d}),\ \vartheta = \mathit{answer}(\mathtt{d})\big\},$$
$$\chi^{\gamma}(\mathsf{X}) := \lambda\mathbf{G}.\ \mathit{wf}_{\mathbf{G}}(\big\{\mathtt{d} \mid \mathit{first}(\mathtt{d}) = \mathbf{G};\ \mathbf{G}\vartheta \leftarrow \mathit{last}(\mathtt{d}) \in \mathsf{X}(\mathbf{G});\ \vartheta = \mathit{answer}(\mathtt{d})\big\}).$$ [5]

The proof that $\chi : \mathbb{C} \to \mathbb{A}_{cr}$ is an observable is analogous to the one given for computed answers in Example 3.1.3 and hence it is omitted. Moreover it can be shown that χ and χ^{γ} satisfy the conditions of Definition 3.3.1. Hence computed resultants is a perfect observable.

Example 3.3.3 (Partial Proof Trees)

Another interesting observable which can be proved to be perfect is the partial proof tree observable. Partial proof trees are used in the construction of the Heyting semantics [66, 67]. We will give now a brief description of partial proof trees. We refer to [67] for further details.

Partial proof trees are represented by equivalence classes of terms formed from atoms and consequence functors, $\Sigma' := \{\vdash, \bot, ', '\}$. $\vdash$ is assumed to be binary non-associative, comma binary left-associative and $\bot$ is a constant. To avoid parenthesis $\vdash$ is assumed to bind tighter than comma. Let V^* be a enumerable set of variables distinct from V which range over partial proof trees. The *Heyting base Hg* is taken to be the lattice completion of the equivalence classes $\mathsf{T}(\Sigma', \mathsf{V}^* \cup \mathit{Atoms})$ modulo variance. We denote by $\bigwedge$ and $\bigvee$ the meet and join operations of the lattice. Moreover $_$ represents an anonymous distinct variable of V^*.

Given a partial proof tree T, by $\mathit{open}(\mathsf{T})$ we denote the list of open subtrees of T, taken from left to right. Furthermore by $\mathit{repl}(\mathsf{T}, \mathsf{T}', \mathsf{T}'')$ we denote the tree which is obtained by replacing the subtree T' of T by T''. For example, if T is the partial proof tree $((_ \vdash \mathsf{C}, \mathit{true} \vdash \mathsf{D}, _ \vdash \mathsf{E}) \vdash \mathsf{A}, _ \vdash \mathsf{B})$, $\mathit{open}(\mathsf{T}) = _ \vdash \mathsf{C}, _ \vdash \mathsf{E}, _ \vdash \mathsf{B}$ and $\mathit{repl}(\mathsf{T}, _ \vdash \mathsf{E}, \mathsf{T}' \vdash \mathsf{F}) = (_ \vdash \mathsf{C}, \mathit{true} \vdash \mathsf{D}, \mathsf{T}' \vdash \mathsf{F}) \vdash \mathsf{A}, _ \vdash \mathsf{B}$.

In order to define the abstraction on the domain $\mathbb{A}_{\mathrm{Hg}} \subseteq [\mathit{Goals} \rightharpoonup \wp(\mathit{Hg})]$ of partial proof tree collections we need to define first the abstraction of goals and clauses.

$$\mathit{Ht}(\mathsf{A}_1, \ldots, \mathsf{A}_n) := _ \vdash \mathsf{A}_1, \ldots, _ \vdash \mathsf{A}_n$$

[5] Recall that $\mathit{wf}_{\mathbf{G}}(\mathsf{S})$ is the (greatest) well-formed subset (of derivations starting from $\mathbf{G}$ only) of any set of derivations S.

$$Ht(\mathsf{H} \leftarrow \mathbf{B}) := Ht(\mathbf{B}) \vdash \mathsf{H}$$

where $Ht(\mathsf{H}\leftarrow) = \mathit{true} \vdash \mathsf{H}$. Then we can define the abstraction $Ht(\mathsf{d})$ of a derivation $\mathsf{d} := \mathbf{G}_0 \xrightarrow[c_1,\ldots,c_n]{\vartheta} \mathbf{G}_n$ by an iterative process.[6] First of all let $\mathsf{T}_0 = Ht(\mathbf{G}_0)$. Then we build any T_i by using T_{i-1} and c_i as follows, $\mathsf{T}_i := \mathit{repl}(\mathsf{T}_{i-1}, \mathit{car}(\mathit{open}(\mathsf{T}_{i-1})), Ht(c_i))$, where the *car* operator selects the first term of a list. The abstraction of the derivation d is then $Ht(\mathsf{d}) = \mathsf{T}_n\vartheta$.

The *partial proof tree* observable $\zeta : \mathbb{C} \to \mathbb{A}_{\mathrm{Hg}}$ is

$$\zeta(\mathsf{D}) = \lambda\mathbf{G}.\left\{ Ht(\mathsf{d}) \mid \mathsf{d} \in \mathsf{D} \right\}.$$

The proof that $\zeta : \mathbb{C} \to \mathbb{A}_{Hg}$ is an observable is analogous to the one given for computed answers in Example 3.1.3 and hence it is omitted. Moreover it can be shown that ζ and ζ^{γ} satisfy the conditions of Definition 3.3.1. Hence partial proof trees is a perfect observable.

Note that there exist observables which are not perfect. For example, the observable ξ of Example 3.1.3 is not a perfect observable, since the axiom (3.3.4) does not hold.

The following theorem shows that the abstract transition relation of perfect observables is precise.

Theorem 3.3.4 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a perfect observable and P be a program. Then $\forall \mathsf{D}, \mathsf{D}' \in \mathbb{C}.\, \mathsf{D} \underset{\mathsf{P}}{\longmapsto}^* \mathsf{D}' \Longrightarrow \alpha(\mathsf{D}) \overset{\alpha}{\underset{\mathsf{P}}{\longmapsto}}^* \alpha(\mathsf{D}')$. Moreover, $\forall \mathsf{X}' \in \mathbb{A}.\, \alpha(\mathsf{D}) \overset{\alpha}{\underset{\mathsf{P}}{\longmapsto}}^* \mathsf{X}' \Longrightarrow \exists \mathsf{D}' \in \mathbb{C}$ such that $\mathsf{X}' = \alpha(\mathsf{D}')$ and $\mathsf{D} \underset{\mathsf{P}}{\longmapsto}^* \mathsf{D}'$.*

Proof. In the following the notation $\bar{\mathsf{D}} \underset{\mathsf{P}}{\longmapsto}^n \mathsf{D}'$ ($\bar{\mathsf{X}} \overset{\alpha}{\underset{\mathsf{P}}{\longmapsto}}^n \mathsf{X}'$) means that the collection $\bar{\mathsf{D}}$ ($\bar{\mathsf{X}}$) results in the collection D' (X') with *at the most* n transition steps $\underset{\mathsf{P}}{\longmapsto}$ ($\overset{\alpha}{\underset{\mathsf{P}}{\longmapsto}}$). We prove the thesis by induction on n.

Base case Straightforward since $\mathsf{D} \underset{\mathsf{P}}{\longmapsto}^0 \mathsf{D}'$ if and only if $\mathsf{D} = \mathsf{D}'$ and $\alpha(\mathsf{D}) \overset{\alpha}{\underset{\mathsf{P}}{\longmapsto}}^0 \mathsf{X}'$ if and only if $\mathsf{X}' = \alpha(\mathsf{D})$.

Inductive case First of all observe that, since α is a perfect observable, for any $\mathsf{D}'' \in \mathbb{C}$,

$$\alpha(\mathsf{D}'' \triangleright su(tree(\mathsf{P}))) = \alpha(\mathsf{D}'') \,\widetilde{\triangleright}\, su_{\alpha}(\alpha(tree(\mathsf{P}))). \tag{1}$$

[6] Partial roof trees are independent from the selection rule. However, since we obtain them by abstracting *SLD*-trees via the leftmost selection rule, we will construct partial proof trees only from left to right.

Now we prove the implication $\Longleftarrow$. The proof of the other implication is analogous and hence it is omitted.

Assume that $\mathsf{D} \underset{\mathsf{P}}{\longmapsto}^{n} \mathsf{D}'$, with $n > 0$. Then, by definition of $\underset{\mathsf{P}}{\longmapsto}^{n}$, there exists $\mathsf{D}'' \in \mathbb{C}$, such that $\mathsf{D} \underset{\mathsf{P}}{\longmapsto}^{n-1} \mathsf{D}'' \underset{\mathsf{P}}{\longmapsto}^{1} \mathsf{D}'' \triangleright su(tree(\mathsf{P}))$. By inductive hypothesis $\alpha(\mathsf{D}) \overset{\alpha}{\underset{\mathsf{P}}{\longmapsto}}^{n-1} \alpha(\mathsf{D}'')$. Therefore, by definition of $\overset{\alpha}{\underset{\mathsf{P}}{\longmapsto}}^{1}$ and (1), $\alpha(\mathsf{D}) \overset{\alpha}{\underset{\mathsf{P}}{\longmapsto}}^{n} \alpha(\mathsf{D}'' \triangleright su(tree(\mathsf{P})))$. ■

We can now show that the operational semantics and the top-down denotation are indeed precise.

Corollary 3.3.5 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a perfect observable, $\mathbf{G}$ be a goal and P be a program. Then*

1. $\alpha(\mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!]) = \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!]$,
2. $\alpha(\mathcal{O}[\![\mathsf{P}]\!]) = \mathcal{O}_\alpha[\![\mathsf{P}]\!]$.

Proof. We prove the points separately.

Point 1

$$\begin{aligned}
&\alpha(\mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!]) = && [\text{by definition of } \mathcal{B}[\![\cdot]\!] \text{ and (3.3.1)}]\\
&\alpha(\textstyle\sum\{\gamma\alpha(\mathsf{D}) \mid \phi_{\mathbf{G}} \underset{\mathsf{P}}{\longmapsto}^{*} \mathsf{D}\}) = && [\text{by Theorem 3.3.4}]\\
&\alpha(\textstyle\sum\{\gamma(\mathsf{X}) \mid \alpha(\phi_{\mathbf{G}}) \overset{\alpha}{\underset{\mathsf{P}}{\longmapsto}}^{*} \mathsf{X}\}) = && [\text{by definition of } \widetilde{\textstyle\sum} \text{ and } \mathcal{B}_\alpha[\![\cdot]\!]]\\
&\mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!].
\end{aligned}$$

Point 2

$$\begin{aligned}
&\alpha(\mathcal{O}[\![\mathsf{P}]\!]) =\\
&\qquad [\text{by definition of } \mathcal{O}[\![\cdot]\!] \text{ and (3.3.1)}]\\
&\alpha(\textstyle\sum\{\gamma\alpha(\mathcal{B}[\![\mathsf{p}(\mathbf{x})\ in\ \mathsf{P}]\!]/_{\equiv_{\mathbb{C}}})\}_{\mathsf{p}(\mathbf{x})\in Goals}) =\\
&\qquad [\text{by Point 3 of Definition 3.1.1}]\\
&\alpha(\textstyle\sum\{(\gamma\alpha(\mathcal{B}[\![\mathsf{p}(\mathbf{x})\ in\ \mathsf{P}]\!]))/_{\equiv_{\mathbb{C}}}\}_{\mathsf{p}(\mathbf{x})\in Goals}) =\\
&\qquad [\text{by definition of } \equiv_{\mathbb{A}}]\\
&\alpha(\textstyle\sum\{\gamma(\alpha(\mathcal{B}[\![\mathsf{p}(\mathbf{x})\ in\ \mathsf{P}]\!])/_{\equiv_{\mathbb{A}}})\}_{\mathsf{p}(\mathbf{x})\in Goals}) =\\
&\qquad [\text{by definition of } \widetilde{\textstyle\sum} \text{ and Point 1}]\\
&\widetilde{\textstyle\sum}\{\mathcal{B}_\alpha[\![\mathsf{p}(\mathbf{x})\ in\ \mathsf{P}]\!]/_{\equiv_{\mathbb{A}}}\}_{\mathsf{p}(\mathbf{x})\in Goals} =\\
&\qquad [\text{by definition of } \mathcal{O}_\alpha[\![\cdot]\!]]
\end{aligned}$$

$\mathcal{O}_\alpha[\![\mathsf{P}]\!]$.

■

We show now that all the properties of *SLD*-trees stated in Chapter 2 hold for the abstract top-down denotation for any perfect observable as well.

Corollary 3.3.6 *Let* $\alpha : \mathbb{C} \to \mathbb{A}$ *be a perfect observable,* A *be an atom,* $\mathbf{G}$, $\mathbf{G}'$ *be goals and* P, P' *be programs. Then*

1. $\mathcal{B}_\alpha[\![\mathsf{A}\ in\ \mathsf{P}]\!] = \mathsf{A} \tilde{\cdot} \mathcal{O}_\alpha[\![\mathsf{P}]\!]$,
2. $\mathcal{B}_\alpha[\![(\mathbf{G}, \mathbf{G}')\ in\ \mathsf{P}]\!] = \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!] \tilde{\times} \mathcal{B}_\alpha[\![\mathbf{G}'\ in\ \mathsf{P}]\!]$,
3. $\mathsf{P} \approx_\alpha \mathsf{P}' \iff \mathcal{O}_\alpha[\![\mathsf{P}]\!] = \mathcal{O}_\alpha[\![\mathsf{P}']\!]$.

Proof. We prove the points separately.

Point 1

$$\begin{array}{ll}
\mathcal{B}_\alpha[\![\mathsf{A}\ in\ \mathsf{P}]\!] = & [\text{by Point 1 of Corollary 3.3.5}] \\
\alpha(\mathcal{B}[\![\mathsf{A}\ in\ \mathsf{P}]\!]) = & [\text{by Point 1 of Theorem 2.5.4 and (3.3.2)}] \\
\alpha(\mathsf{A} \cdot \gamma\alpha(\mathcal{O}[\![\mathsf{P}]\!])) = & [\text{by Point 2 of Corollary 3.3.5}] \\
\alpha(\mathsf{A} \cdot \gamma(\mathcal{O}_\alpha[\![\mathsf{P}]\!])) = & [\text{by definition of } \tilde{\cdot}] \\
\mathsf{A} \tilde{\cdot} \mathcal{O}_\alpha[\![\mathsf{P}]\!]. &
\end{array}$$

Point 2 Analogous to the previous one and hence omitted.

Point 3 By (3.1.1) and Point 1 of Corollary 3.3.5,

$$\mathsf{P} \approx_\alpha \mathsf{P}' \iff \forall \mathbf{G} \in \mathit{Goals}.\ \alpha(\mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!]) = \alpha(\mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}']\!]) \iff$$
$$\forall \mathbf{G} \in \mathit{Goals}.\ \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!] = \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}']\!].$$

Now the proof is analogous to the one of Corollary 2.5.6 (on page 64). By definition of $\mathcal{O}_\alpha[\![\cdot]\!]$, the minimality is trivial. The proof of the converse is by contradiction, by using Points 1 and 2 and by structural induction on the goal $\mathbf{G}$, such that $\mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!] \neq \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}']\!]$. ■

In order to express the abstract *OR*-compositionality we have to define the abstract version $\tilde{\uplus}$ of the $\uplus$ operator. Given $\mathsf{X}_1, \mathsf{X}_2 \in \mathbb{PA}$,

$$\mathsf{X}_1 \tilde{\uplus} \mathsf{X}_2 := [\mathsf{X}_1 \tilde{+} \mathsf{X}_2]^*_\alpha, \tag{3.3.5}$$

where $[\mathsf{X}]^*_\alpha$ is the least solution of the equation

$$[\mathsf{X}]^*_\alpha = \alpha(Id_{\mathbb{I}}) \tilde{+} ([\mathsf{X}]^*_\alpha \tilde{\rhd} su_\alpha(\mathsf{X})), \tag{3.3.6}$$

or (equivalently) the least fixpoint of the continuous operator

$$\widetilde{\mathcal{H}}_{\mathsf{X}}(\mathsf{X}') := \alpha(Id_{\mathbb{I}}) \mathbin{\widetilde{+}} \mathsf{X}' \mathbin{\widetilde{\triangleright}} su_{\alpha}(\mathsf{X}). \tag{3.3.7}$$

Lemma 3.3.7 *Let* $\alpha : \mathbb{C} \to \mathbb{A}$ *be a perfect observable and let* $\mathsf{D} \in \mathbb{PC}$. *Then* $[\alpha(\mathsf{D})]^*_{\alpha} = \alpha([\mathsf{D}]^*)$.

Proof. Let $\mathsf{X} \in \mathbb{PA}$. Now we prove that $\widetilde{\mathcal{H}}_{\alpha(\mathsf{D})} \circ \alpha = \alpha \circ \mathcal{H}_{\mathsf{D}}$.

$$\begin{aligned}
&\widetilde{\mathcal{H}}_{\alpha(\mathsf{D})}(\alpha(\mathsf{D}')) = \\
&\qquad [\text{by definition of } \widetilde{\mathcal{H}}_{\alpha(\mathsf{D})}] \\
&\alpha(Id_{\mathbb{I}}) \mathbin{\widetilde{+}} \alpha(\mathsf{D}') \mathbin{\widetilde{\triangleright}} su_{\alpha}(\alpha(\mathsf{D})) = \\
&\qquad [\text{by definition of } \widetilde{+}, \widetilde{\triangleright} \text{ and since } su_{\alpha} = \alpha \circ su \circ \gamma] \\
&\alpha(\gamma\alpha(Id_{\mathbb{I}}) + \gamma\alpha(\gamma\alpha(\mathsf{D}') \triangleright \gamma\alpha(su(\gamma\alpha(\mathsf{D}))))) = \\
&\qquad [\text{by Definition 3.3.1}] \\
&\alpha(Id_{\mathbb{I}} + \mathsf{D}' \triangleright su(\mathsf{D})) = \\
&\qquad [\text{by definition of } \mathcal{H}_{\mathsf{D}}] \\
&\alpha(\mathcal{H}_{\mathsf{D}}(\mathsf{D}')).
\end{aligned}$$

Now, since $\bot_{\mathbb{A}} = \alpha(\bot_{\mathbb{C}})$ and by a straightforward inductive argument, for any $n \geq 0$, $\widetilde{\mathcal{H}}_{\alpha(\mathsf{D})}{\uparrow}n = \alpha(\mathcal{H}_{\mathsf{D}}{\uparrow}n)$. Then $[\alpha(\mathsf{D})]^*_{\alpha} = \mathit{lfp}_{\mathbb{A}}\, \widetilde{\mathcal{H}}_{\alpha(\mathsf{D})} = \alpha(\mathit{lfp}_{\mathbb{C}}\, \mathcal{H}_{\mathsf{D}}) = \alpha([\mathsf{D}]^*)$. ∎

Corollary 3.3.8 *Let* $\alpha : \mathbb{C} \to \mathbb{A}$ *be a perfect observable and* P_1, P_2 *be programs. Then* $\mathcal{O}_{\alpha}[\![\mathsf{P}_1 \cup \mathsf{P}_2]\!] = \mathcal{O}_{\alpha}[\![\mathsf{P}_1]\!] \mathbin{\widetilde{\uplus}} \mathcal{O}_{\alpha}[\![\mathsf{P}_2]\!]$.

Proof. The following equivalences hold.

$$\begin{aligned}
&\mathcal{O}_{\alpha}[\![\mathsf{P}_1]\!] \mathbin{\widetilde{\uplus}} \mathcal{O}_{\alpha}[\![\mathsf{P}_2]\!] = && [\text{by definition of } \widetilde{\uplus}] \\
&[\mathcal{O}_{\alpha}[\![\mathsf{P}_1]\!] \mathbin{\widetilde{+}} \mathcal{O}_{\alpha}[\![\mathsf{P}_2]\!]]^*_{\alpha} = && [\text{by Corollary 3.3.5 and by (3.3.1)}] \\
&[\alpha(\mathcal{O}[\![\mathsf{P}_1]\!] + \mathcal{O}[\![\mathsf{P}_2]\!])]^*_{\alpha} = && [\text{by Lemma 3.3.7}] \\
&\alpha([\mathcal{O}[\![\mathsf{P}_1]\!] + \mathcal{O}[\![\mathsf{P}_2]\!]]^*) = && [\text{by definition of } \uplus] \\
&\alpha(\mathcal{O}[\![\mathsf{P}_1]\!] \uplus \mathcal{O}[\![\mathsf{P}_2]\!]) = && [\text{by Theorem 2.5.9}] \\
&\alpha(\mathcal{O}[\![\mathsf{P}_1 \cup \mathsf{P}_2]\!]) = && [\text{by Corollary 3.3.5}] \\
&\mathcal{O}_{\alpha}[\![\mathsf{P}_1 \cup \mathsf{P}_2]\!].
\end{aligned}$$

∎

The following theorem shows that the abstract denotational semantics and the bottom-up denotation are precise.

Theorem 3.3.9 *Let* $\alpha : \mathbb{C} \to \mathbb{A}$ *be a perfect observable,* c *be a clause,* A *be an atom,* $\mathbf{G}$ *be a goal and* P *be a program. Then*

1. $\alpha(\mathcal{A}[\![\mathsf{A}]\!]_{\mathrm{I}}) = \mathcal{A}_\alpha[\![\mathsf{A}]\!]_{\alpha(\mathrm{I})}$,
2. $\alpha(\mathcal{G}[\![\mathbf{G}]\!]_{\mathrm{I}}) = \mathcal{G}_\alpha[\![\mathbf{G}]\!]_{\alpha(\mathrm{I})}$,
3. $\alpha(\mathcal{C}[\![\mathsf{c}]\!]_{\mathrm{I}}) = \mathcal{C}_\alpha[\![\mathsf{c}]\!]_{\alpha(\mathrm{I})}$,
4. $\alpha(\mathcal{P}[\![\mathsf{P}]\!]_{\mathrm{I}}) = \mathcal{P}_\alpha[\![\mathsf{P}]\!]_{\alpha(\mathrm{I})}$,
5. $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ *is continuous on* $\mathbb{A}$ *and* $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow}\omega$,
6. $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!]$,
7. $\alpha(\mathcal{Q}[\![\mathbf{G} \textit{ in } \mathsf{P}]\!]) = \mathcal{Q}_\alpha[\![\mathbf{G} \textit{ in } \mathsf{P}]\!]$.

Proof. We prove the points separately.

Point 1 By definition of $\mathcal{A}[\![\cdot]\!]$, $\tilde{\cdot}$, $\mathcal{A}_\alpha[\![\cdot]\!]$ and by (3.3.2), $\alpha(\mathcal{A}[\![\mathsf{A}]\!]_{\mathrm{I}}) = \alpha(\mathsf{A} \cdot \mathrm{I}) = \alpha(\mathsf{A} \cdot \gamma\alpha(\mathrm{I})) = \mathsf{A} \tilde{\cdot} \alpha(\mathrm{I}) = \mathcal{A}_\alpha[\![\mathsf{A}]\!]_{\alpha(\mathrm{I})}$.

Point 2 The proof is by induction on $\mathbf{G}$. If $\mathbf{G} = \square$, by definition of $\mathcal{G}[\![\cdot]\!]$ and $\mathcal{G}_\alpha[\![\cdot]\!]$, $\alpha(\mathcal{G}[\![\square]\!]_{\mathrm{I}}) = \alpha(\phi_\square) = \mathcal{G}_\alpha[\![\square]\!]_{\alpha(\mathrm{I})}$.

Otherwise let $\mathbf{G} = (\mathsf{A}, \mathbf{G}')$. The following equivalences hold.

$$
\begin{array}{ll}
\alpha(\mathcal{G}[\![(\mathsf{A}, \mathbf{G}')]\!]_{\mathrm{I}}) = & [\text{by definition of } \mathcal{G}[\![\cdot]\!]] \\
\alpha(\mathcal{A}[\![\mathsf{A}]\!]_{\mathrm{I}} \times \mathcal{G}[\![\mathbf{G}']\!]_{\mathrm{I}}) = & [\text{by (3.3.3)}] \\
\alpha(\gamma\alpha(\mathcal{A}[\![\mathsf{A}]\!]_{\mathrm{I}}) \times \gamma\alpha(\mathcal{G}[\![\mathbf{G}']\!]_{\mathrm{I}})) = & [\text{by definition of } \tilde{\times}] \\
\alpha(\mathcal{A}[\![\mathsf{A}]\!]_{\mathrm{I}}) \tilde{\times} \alpha(\mathcal{G}[\![\mathbf{G}']\!]_{\mathrm{I}}) = & [\text{by Point 1 and by inductive hypothesis}] \\
\mathcal{A}_\alpha[\![\mathsf{A}]\!]_{\alpha(\mathrm{I})} \tilde{\times} \mathcal{G}_\alpha[\![\mathbf{G}']\!]_{\alpha(\mathrm{I})} = & [\text{by definition of } \mathcal{G}_\alpha[\![\cdot]\!]] \\
\mathcal{G}_\alpha[\![(\mathsf{A}, \mathbf{G}')]\!]_{\alpha(\mathrm{I})}. &
\end{array}
$$

Point 3 Let $\mathsf{c} = \mathsf{H} \leftarrow \mathbf{B}$. Then

$$
\begin{array}{ll}
\alpha(\mathcal{C}[\![\mathsf{c}]\!]_{\mathrm{I}}) = & [\text{by definition of } \mathcal{C}[\![\cdot]\!]] \\
\alpha(\mathit{tree}(\mathsf{c}) \triangleright \mathcal{G}[\![\mathbf{B}]\!]_{\mathrm{I}}) = & [\text{by (3.3.4)}] \\
\alpha(\gamma\alpha(\mathit{tree}(\mathsf{c})) \triangleright \gamma\alpha(\mathcal{G}[\![\mathbf{B}]\!]_{\mathrm{I}})) = & [\text{by definition of } \tilde{\triangleright}] \\
\alpha(\mathit{tree}(\mathsf{c})) \tilde{\triangleright} \alpha(\mathcal{G}[\![\mathbf{B}]\!]_{\mathrm{I}}) = & [\text{by Point 2 and by definition of } \mathcal{C}_\alpha[\![\cdot]\!]] \\
\mathcal{C}_\alpha[\![\mathsf{c}]\!]_{\alpha(\mathrm{I})}. &
\end{array}
$$

Point 4 Immediate by definition of $\mathcal{P}[\![\cdot]\!]$, $\mathcal{P}_\alpha[\![\cdot]\!]$ and by Point 3.

Point 5 Let $\{X_i\}_{i\in I} \subseteq \mathbb{A}$ be a chain. Since $\widetilde{\sum}$ is the *lub* operation on $\mathbb{A}$, we have to prove that $\widetilde{\sum}\{\mathcal{P}_\alpha[\![P]\!]_{X_i}\}_{i\in I} = \mathcal{P}_\alpha[\![P]\!]_{\widetilde{\sum}\{X_i\}_{i\in I}}$.

$$\begin{array}{ll}
\widetilde{\sum}\{\mathcal{P}_\alpha[\![P]\!]_{X_i}\}_{i\in I} = & [\text{by definition of } \widetilde{\sum}] \\
\alpha(\sum\{\gamma(\mathcal{P}_\alpha[\![P]\!]_{X_i})\}_{i\in I}) = & [\text{since } X_i = \alpha\gamma(X_i) \text{ and by Point 4}] \\
\alpha(\sum\{\gamma\alpha(\mathcal{P}[\![P]\!]_{\gamma(X_i)})\}_{i\in I}) = & [\text{by (3.3.1)}] \\
\alpha(\sum\{\mathcal{P}[\![P]\!]_{\gamma(X_i)}\}_{i\in I}) = & [\text{since } \mathcal{P}[\![P]\!] \text{ is continuous}] \\
\alpha(\mathcal{P}[\![P]\!]_{\sum\{\gamma(X_i)\}_{i\in I}}) = & [\text{by Point 4 and definition of } \widetilde{\sum}] \\
\mathcal{P}_\alpha[\![P]\!]_{\widetilde{\sum}\{X_i\}_{i\in I}}. &
\end{array}$$

Then apply Tarsky's theorem.

Point 6 First of all note that, since $(\mathbb{A}, \leq)$ is a complete lattice, there exists the bottom element $\bot_{\mathbb{A}}$. Moreover, since α is monotonic and ϕ is the bottom element of $\mathbb{C}$, for any $D \in \mathbb{C}$, $\alpha(\phi) \leq \alpha(D)$ and then, since α is surjective, $\alpha(\phi) = \bot_{\mathbb{A}}$. Then, by Point 3 and a straightforward inductive argument, for any $n \geq 0$, $\alpha(\mathcal{P}[\![P]\!]\uparrow n) = \mathcal{P}_\alpha[\![P]\!]\uparrow n$. Therefore, since $\sum$ is the *lub* operation on $\mathbb{C}$ and $\widetilde{\sum}$ is the *lub* operation on $\mathbb{A}$,

$$\begin{array}{ll}
\alpha(\mathcal{F}[\![P]\!]) = & [\text{by definition of } \mathcal{F}[\![\cdot]\!]] \\
\alpha(\mathcal{P}[\![P]\!]\uparrow\omega) = & [\text{since } \mathcal{P}[\![P]\!] \text{ is continuous}] \\
\alpha(\sum\{\mathcal{P}[\![P]\!]\uparrow n\}_{n\geq 0}) = & [\text{by (3.3.1) and definition of } \widetilde{\sum}] \\
\widetilde{\sum}\{\alpha(\mathcal{P}[\![P]\!]\uparrow n)\}_{n\geq 0} = & [\text{by the previous observation}] \\
\widetilde{\sum}\{\mathcal{P}_\alpha[\![P]\!]\uparrow n\}_{n\geq 0} = & [\text{by Point 5}] \\
\mathcal{F}_\alpha[\![P]\!]\,. &
\end{array}$$

Point 7 By definition of $\mathcal{Q}[\![\cdot]\!]$, $\mathcal{Q}_\alpha[\![\cdot]\!]$ and by Points 2 and 6, $\alpha(\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ P]\!]) = \alpha(\mathcal{G}[\![\mathbf{G}]\!]_{\mathit{lfp}\,\mathcal{P}[\![P]\!]}) = \mathcal{G}_\alpha[\![\mathbf{G}]\!]_{\alpha(\mathit{lfp}\,\mathcal{P}[\![P]\!])} = \mathcal{G}_\alpha[\![\mathbf{G}]\!]_{\mathit{lfp}\,\mathcal{P}_\alpha[\![P]\!]} = \mathcal{Q}_\alpha[\![\mathbf{G}\ \mathit{in}\ P]\!]$. ■

Finally, by using Theorem 3.3.9 and Corollaries 3.3.6 and 3.3.5, we can prove the equivalences between the denotational and the operational semantics on one side, and between the top-down and bottom-up denotations on the other side.

Corollary 3.3.10 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a perfect observable, $\mathbf{G}$ be a goal and P, P' be programs. Then*

1. $\mathcal{O}_\alpha[\![P]\!] = \mathcal{F}_\alpha[\![P]\!]$,

2. $\mathcal{Q}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!] = \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!]$,

3. $\mathsf{P} \approx_\alpha \mathsf{P}' \Longleftrightarrow \mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathcal{F}_\alpha[\![\mathsf{P}']\!]$.

There are several examples of interesting observables for which $\widetilde{\triangleright}$ is not precise (for example — as already mentioned — computed answers). Due to the imprecision of the low level operations, we can still try to define a more precise semantics by choosing the optimal abstractions for an upper level semantic operation. In the denotational semantics, the operator $\triangleright$ is used only to define the semantic function $\mathcal{C}[\![\cdot]\!]$. In the Section 3.4 we obtain a new class of observables by taking its optimal abstraction $\widetilde{\mathcal{C}}[\![\cdot]\!]$.

3.3.1 Computed Resultant Semantics

We show now how to reconstruct the semantics modeling computed resultants, defined in [54], by means of the observable χ of Example 3.3.2.

In Section 3.9 we prove that, by applying (3.2.1), (3.2.2), (3.2.3) and (3.2.4), the $\widetilde{\cdot}$, $\widetilde{\triangleright}$ and $\widetilde{\times}$ operations are

$$\begin{aligned}
A \widetilde{\cdot} X &= \phi\left[{}^{R}/_{A}\right] \text{ where } R := \big\{(A \leftarrow \mathbf{B}')\vartheta \mid \langle H,\, R'\rangle \text{ is a renamed} \\
&\qquad \text{apart (from } A\text{) version of } \langle A',\, X(A')\rangle, \text{ for some } A' \leq A, \\
&\qquad H' \leftarrow \mathbf{B}' \in R', \text{ there exists } \delta \text{ s.t. } A = H\delta \text{ and } \vartheta = mgu(A, H')\big\}, \\
X_1 \widetilde{\times} X_2 &= \lambda\mathbf{G}.\,\big\{((\mathbf{G}_1, \mathbf{G}_2) \leftarrow \mathbf{B})\vartheta \mid \mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2), \forall i = 1, 2, \mathbf{G}'_i \equiv \mathbf{G}_i, \\
&\qquad r_i = \mathbf{G}''_i \leftarrow \mathbf{B}_i \text{ is a renamed version of an element in } X_i(\mathbf{G}'_i), \text{ via a} \\
&\qquad \text{renaming } \rho_i \text{ s.t. } \mathbf{G}'_i\rho_i = \mathbf{G}_i,\ var(r_1) \cap var(r_2) \subseteq var(\mathbf{G}_1) \cap \\
&\qquad var(\mathbf{G}_2),\ \mathbf{G}_1\vartheta_1 = \mathbf{G}''_1, \text{ and if } \mathbf{B}_1 \neq \square \text{ then } \vartheta = \vartheta_1|_{\mathbf{G}_1},\ \mathbf{B} = (\mathbf{B}_1, \mathbf{G}_2) \\
&\qquad \text{else } \mathbf{B} = \mathbf{B}_2,\ \vartheta = \vartheta_1|_{\mathbf{G}_1} \circ mgu(\mathbf{G}_2\vartheta_1, \mathbf{G}''_2)\big\}, \\
X_1 \widetilde{\triangleright} X_2 &= \lambda\mathbf{G}.\, X_1(\mathbf{G}) \cup \big\{(\mathbf{G}' \leftarrow \mathbf{G}_3)\vartheta \mid\ r_1 = \mathbf{G}' \leftarrow \mathbf{G}_1 \in X_1(\mathbf{G}), \\
&\qquad \mathbf{G}_1 \equiv \mathbf{G}_2,\ r_2 = \mathbf{G}'_2 \leftarrow \mathbf{G}_3 \text{ is a renamed version of an element in} \\
&\qquad X_2(\mathbf{G}_2), \text{ via a renaming } \rho \text{ s.t. } \mathbf{G}_2\rho = \mathbf{G}_1, \\
&\qquad var(r_1) \cap var(r_2) \subseteq var(\mathbf{G}_1),\ \mathbf{G}_1\vartheta = \mathbf{G}'_2 \text{ and } dom(\vartheta) \subseteq var(\mathbf{G}_1)\big\}.
\end{aligned}$$

while the optimal $\widetilde{\sum}$ operation turns out to be point-wise union.

Example 3.3.11

Consider the following computed resultants collections.

$$X_1 := \begin{cases} p(x,y) \mapsto \{p(x,y) \leftarrow p(x,y);\ \ p(x,y) \leftarrow q(x,y);\ \ p(g(a),b) \leftarrow \square\} \\ n(x) \quad \mapsto \{n(x) \leftarrow n(x);\ \ n(x) \leftarrow p(x,x)\} \end{cases}$$

$$X_2 := \begin{cases} q(x,y) \quad \mapsto W_1 \\ q(f(x),z) \mapsto W_2 \end{cases}$$

where

$$W_1 := \{q(x,y) \leftarrow q(x,y);\ \ q(x,y) \leftarrow n(f(s),x);\ \ q(a,y) \leftarrow \square\}$$
$$W_2 := \{q(f(x),z) \leftarrow q(f(x),z);\ \ q(f(x),a) \leftarrow m(g(s),f(x));$$
$$q(f(g(s)),a) \leftarrow \square\}$$

The following hold.

$$q(f(g(z)),z) \tilde{\cdot} X_2 = q(f(g(z)),z) \mapsto W_3$$
$$X_1 \tilde{\triangleright} X_2 = \begin{cases} p(x,y) \mapsto X_1(p(x,y)) \cup W_4 \\ n(x) \quad \mapsto X_1(n(x)) \end{cases}$$
$$X_1 \tilde{\times} X_2 = \begin{cases} p(x,y), q(x,y) & \mapsto W_5 \\ p(x,y), q(f(x),y) & \mapsto W_6 \\ n(x), q(x,y) & \mapsto W_7 \\ n(x), q(f(x),y) & \mapsto W_8 \end{cases}$$

where

$$W_3 := \{q(f(g(z)),z) \leftarrow q(f(g(z)),z);\ \ q(f(g(z)),z) \leftarrow n(f(s),f(g(z)));$$
$$q(f(g(a)),a) \leftarrow m(g(s),f(g(a)));\ \ q(f(g(a)),a) \leftarrow \square\}$$
$$W_4 := \{p(x,y) \leftarrow n(f(s),x);\ \ p(a,y) \leftarrow \square\}$$
$$W_5 := \{p(x,y), q(x,y) \leftarrow p(x,y), q(x,y);\ \ p(x,y), q(x,y) \leftarrow q(x,y), q(x,y);$$
$$p(g(a),b), q(g(a),b) \leftarrow q(g(a),b);$$
$$p(g(a),b), q(g(a),b) \leftarrow n(f(s),g(a))\}$$
$$W_6 := \{p(x,y), q(f(x),y) \leftarrow p(x,y), q(f(x),y);$$
$$p(x,y), q(f(x),z) \leftarrow q(x,y), q(f(x),z);$$
$$p(g(a),b), q(f(g(a)),z) \leftarrow q(f(g(a)),z);$$
$$p(g(a),b), q(f(g(a)),a) \leftarrow m(g(s),f(g(a)));$$
$$p(g(a),b), q(f(g(a)),a) \leftarrow \square\}$$
$$W_7 := \{n(x), q(x,y) \leftarrow n(x), q(x,y);\ \ n(x), q(x,y) \leftarrow p(x,x), q(x,y)\}$$
$$W_8 := \{n(x), q(f(x),y) \leftarrow n(x), q(f(x),y);$$
$$n(x), q(f(x),z) \leftarrow p(x,x), q(f(x),z)\}$$

Note that collection X_1 contains the resultant $p(x,y) \leftarrow q(x,y)$ and then the $\tilde{\triangleright}$ can use the resultants for $q(x,y)$ of X_2 to extend $X_1(p(x,y))$ with W_4.

The abstract semantic function is

$$\begin{aligned}\mathcal{C}_\chi[\![p(\mathbf{t}) \leftarrow \mathbf{B}]\!]_X = \lambda p(\mathbf{x}).\, \big\{ & p(\mathbf{x})\vartheta \leftarrow (\mathbf{B}_k, \mathbf{B}'') \mid \mathbf{B} = (\mathbf{B}', \mathbf{B}''),\ \mathbf{x} \text{ are new} \\ & \text{variables, } \exists k \text{ s.t. } \forall i < k.\, A_i \leftarrow \square \in X(p_i(\mathbf{x}_i)), \\ & A_k \leftarrow \mathbf{B}_k \in X(p_k(\mathbf{x}_k)),\ X(p_j(\mathbf{x}_j)) \neq \emptyset \text{ is defined for any} \\ & p_j \in \mathit{preds}(\mathbf{B}''),\ \vartheta := \{\mathbf{x}/\mathbf{t}\} \circ \vartheta', \\ & \vartheta' := \mathit{mgu}\big(\mathbf{B}', (A_1, \ldots, A_k)\big)\ \big\}.\end{aligned}$$

The abstract top-down denotation is

$$\mathcal{O}_\chi[\![P]\!] = \lambda p(\mathbf{x}).\, \big\{ p(\mathbf{x})\vartheta \leftarrow \mathbf{B} \mid p(\mathbf{x}) \xrightarrow[P]{\vartheta}{}^{*}\ \mathbf{B} \big\} \big/_{\equiv_{\mathbb{A}_{cr}}}$$

and coincides with the abstract bottom-up denotation and the abstraction of the top-down denotation, i.e., $\mathcal{O}_\chi[\![P]\!] = \mathcal{F}_\chi[\![P]\!] = \chi(\mathcal{O}[\![P]\!])$.

3.3.2 The Partial Proof Tree Semantics

We show now how to obtain the semantics modeling partial proof trees, by means of the observable ζ of Example 3.3.3.

We first need to introduce some notation to get a compact presentation of the operations. Given a tree T, we denote by $\langle T \rangle^n_m$ the tree obtained by adding n anonymous tree variables to the left of T and m anonymous tree variables to the right. Moreover $\mathit{prfx}(T)$ is the list (in the reverse order) of all trees which are obtained starting from T and then by sequentially replacing a subtree by an anonymous tree variable going form right to left w.r.t. comma and form left to right w.r.t. $\vdash$. Intuitively, *prfx* gives the history of all the prefixes of T built in a left to right order. For example, if $T := (_ \vdash A \vdash B, _ \vdash C)$, $\mathit{prfx}(T) = [(_, _); (_ \vdash B, _); (_ \vdash A \vdash B, _); T]$.

By applying (3.2.1), (3.2.2), (3.2.3) and (3.2.4), the $\widetilde{\cdot}$, $\widetilde{\rhd}$ and $\widetilde{\times}$ operations are

$$\begin{aligned}A \,\widetilde{\cdot}\, X &= \phi\big[{}^{S}/_{A}\big] \text{ where } S := \big\{(_ \vdash A) \wedge T \mid T \in X(H), H \geq A\big\}, \\ X_1 \,\widetilde{\times}\, X_2 &= \lambda \mathbf{G}.\, \big\{ T \mid (\mathbf{G}', \mathbf{G}'') = \mathbf{G}, T' \in X_1(\mathbf{G}'), T'' \in X_2(\mathbf{G}''), \\ & \qquad \mathbf{G}' = A_1, \ldots, A_n,\ \mathbf{G}'' = B_1, \ldots, B_m, \\ & \qquad T_1, \ldots, T_k = (\mathit{prfx}(\langle T' \rangle^0_m), \mathit{prfx}(\langle T'' \rangle^n_0)), \\ & \qquad T = \bigvee_{1 \leq i \leq k} \left(\mathit{Ht}(\mathbf{G}) \wedge \bigwedge_{1 \leq j \leq i} T_j \right) \big\}, \\ X_1 \,\widetilde{\rhd}\, X_2 &= \lambda \mathbf{G}.\, X_1(\mathbf{G}) \cup \big\{ T \wedge \bigwedge_{1 \leq i \leq n} \mathit{repl}(T, _ \vdash A_i, T_i)) \mid T \in X_1(\mathbf{G}), \\ & \qquad \mathit{open}(T) = _ \vdash A_1, \ldots, _ \vdash A_n,\ (T_1, \ldots, T_n) \in X_2(A_1, \ldots, A_n) \big\}.\end{aligned}$$

while the optimal $\widetilde{\sum}$ operation turns out to be point-wise union.

The abstract semantic function is

$$\mathcal{P}_\zeta[\![\mathsf{P}]\!]_\mathsf{X} = \lambda p(\mathbf{x}).\,\big\{(_ \vdash p(\mathbf{x})) \wedge Ht(c) \wedge (\mathsf{T} \vdash p(\mathbf{t})) \mid \mathsf{c} = p(\mathbf{t}) \leftarrow \mathsf{B}_1, \ldots, \mathsf{B}_n \in \mathsf{P},$$
$$\mathsf{T} = \bigvee_{1 \le i \le k} \left(Ht(\mathsf{B}_1, \ldots, \mathsf{B}_n) \wedge \bigwedge_{1 \le j \le i} \mathsf{T}_j \right),\ \mathsf{T}_1, \ldots, \mathsf{T}_k =$$
$$(prfx(\langle \mathsf{X}(\mathsf{B}_1)\rangle_{n-1}^{0}), prfx(\langle \mathsf{X}(\mathsf{B}_2)\rangle_{n-2}^{1}), \ldots, prfx(\langle \mathsf{X}(\mathsf{B}_n)\rangle_{0}^{n-1}))\big\}.$$

Since ζ is perfect, the abstract bottom-up denotation $\mathcal{F}_\zeta[\![\mathsf{P}]\!]$, the abstract top-down denotation $\mathcal{O}_\zeta[\![\mathsf{P}]\!]$ and the abstraction of the top-down denotation $\zeta(\mathcal{O}[\![\mathsf{P}]\!])$ coincide.

Let us finally note that the *Heyting semantics* [66, 67] can be obtained by collecting from $\mathcal{O}_\zeta[\![\mathsf{P}]\!]$ all complete proof trees (trees which do not contain anonymous tree variables). An observable α which model complete proof trees can be easily defined using a construction analogous to this one, but it is *not* anymore perfect, yet it is denotational (denotational observables are introduced in the next section). It turns out that its $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ operator is isomorphic to the HgT_P operator of [67] and that the bottom-up denotation $\mathcal{F}_\alpha[\![\mathsf{P}]\!]$ models the Heyting semantics of P.

3.4 Denotational Observables

We relax the optimality condition of axiom (3.3.4) and admit a $\widetilde{\triangleright}$ operator which is not necessarily precise.

Definition 3.4.1 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be an observable. Then α is a* denotational observable *if*

$$\alpha(\mathsf{A} \cdot \mathsf{D}) = \alpha(\mathsf{A} \cdot (\gamma \circ \alpha)\mathsf{D}), \tag{3.4.1}$$
$$\alpha(\mathsf{D} \times \mathsf{D}') = \alpha((\gamma \circ \alpha)\mathsf{D} \times (\gamma \circ \alpha)\mathsf{D}'), \tag{3.4.2}$$
$$\alpha(\mathsf{D} \triangleright \mathsf{D}') = \alpha(\mathsf{D} \triangleright (\gamma \circ \alpha)\mathsf{D}'). \tag{3.4.3}$$

The following theorems show that, under these conditions, we can just replace $\mathcal{C}_\alpha[\![\cdot]\!]$ by the optimal abstraction $\widetilde{\mathcal{C}}[\![\cdot]\!]$ of $\mathcal{C}[\![\cdot]\!]$ to make the semantic definition precise[7].

$$\widetilde{\mathcal{C}}[\![\mathsf{c}]\!] := \alpha \circ \mathcal{C}[\![\mathsf{c}]\!] \circ \gamma. \tag{3.4.4}$$

With this new semantic operator we can define a more precise denotational semantics, simply by replacing equation (3.2.11) with

$$\mathcal{P}_\alpha[\![\{\mathsf{c}\} \cup \mathsf{P}]\!]_\mathsf{X} := \widetilde{\mathcal{C}}[\![\mathsf{c}]\!]_\mathsf{X} \mathbin{\widetilde{+}} \mathcal{P}_\alpha[\![\mathsf{P}]\!]_\mathsf{X}. \tag{3.4.5}$$

Then, if we replace $\mathcal{C}_\alpha[\![\cdot]\!]$ by $\widetilde{\mathcal{C}}[\![\cdot]\!]$ in its statement, Theorem 3.3.9 also holds for denotational observables.

[7] Note that if a denotational observable α is also perfect, then $\widetilde{\mathcal{C}}[\![\cdot]\!] = \mathcal{C}_\alpha[\![\cdot]\!]$.

Theorem 3.4.2 *Let* $\alpha : \mathbb{C} \to \mathbb{A}$ *be a denotational observable,* c *be a clause,* A *be an atom,* $\mathbf{G}$ *be a goal and* P *be a program. Then*

1. $\alpha(\mathcal{A}[\![\mathsf{A}]\!]_{\mathrm{I}}) = \mathcal{A}_\alpha[\![\mathsf{A}]\!]_{\alpha(\mathrm{I})}$,
2. $\alpha(\mathcal{G}[\![\mathbf{G}]\!]_{\mathrm{I}}) = \mathcal{G}_\alpha[\![\mathbf{G}]\!]_{\alpha(\mathrm{I})}$,
3. $\alpha(\mathcal{C}[\![\mathsf{c}]\!]_{\mathrm{I}}) = \widetilde{\mathcal{C}}[\![\mathsf{c}]\!]_{\alpha(\mathrm{I})}$,
4. $\alpha(\mathcal{P}[\![\mathsf{P}]\!]_{\mathrm{I}}) = \mathcal{P}_\alpha[\![\mathsf{P}]\!]_{\alpha(\mathrm{I})}$,
5. $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ *is continuous on* $\mathbb{A}$ *and* $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathcal{P}_\alpha[\![\mathsf{P}]\!]\uparrow\omega$,
6. $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!]$ *and* $\alpha(\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}]\!]) = \mathcal{Q}_\alpha[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}]\!]$.

Proof. We prove only Point 3. The proof of the other statements is analogous to those of Theorem 3.3.9 (by using Definition 3.4.1 and the definition of $\widetilde{\mathcal{C}}[\![\cdot]\!]$ instead of Definition 3.3.1 and the definition of $\mathcal{C}_\alpha[\![\cdot]\!]$, respectively) and therefore it is omitted. Let c be the clause $\mathsf{H} \leftarrow \mathbf{B}$. Then

$$
\begin{array}{ll}
\alpha(\mathcal{C}[\![\mathsf{c}]\!]_{\mathrm{I}}) = & [\text{by definition of } \mathcal{C}[\![\cdot]\!] \text{ and by (3.4.3)}] \\
\alpha(\mathit{tree}(\mathsf{c}) \triangleright \gamma\alpha(\mathcal{G}[\![\mathbf{B}]\!]_{\mathrm{I}})) = & [\text{by Point 2}] \\
\alpha(\mathit{tree}(\mathsf{c}) \triangleright \gamma(\mathcal{G}_\alpha[\![\mathbf{B}]\!]_{\alpha(\mathrm{I})})) = & [\text{since } \mathrm{I} = \alpha\gamma(\mathrm{I}) \text{ and by Point 2}] \\
\alpha(\mathit{tree}(\mathsf{c}) \triangleright \gamma\alpha(\mathcal{G}[\![\mathbf{B}]\!]_{\gamma\alpha(\mathrm{I})})) = & [\text{by (3.4.3) and by definition of } \widetilde{\mathcal{C}}[\![\cdot]\!]] \\
\widetilde{\mathcal{C}}[\![\mathsf{c}]\!]_{\alpha(\mathrm{I})}. &
\end{array}
$$

■

As a consequence of the above theorem, the abstract denotational semantics and the bottom-up denotation are precise. In particular, since $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \alpha(\mathcal{F}[\![\mathsf{P}]\!])$, $\mathcal{F}_\alpha[\![\cdot]\!]$ is correct and minimal w.r.t. α. Remember that the *AND*-compositionality property of $\mathcal{Q}_\alpha[\![\cdot]\!]$ follows by construction.

Corollary 3.4.3 *Let* $\alpha : \mathbb{C} \to \mathbb{A}$ *be a denotational observable and* P, P' *be programs. Then* $\mathsf{P} \approx_\alpha \mathsf{P}' \iff \mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathcal{F}_\alpha[\![\mathsf{P}']\!]$.

Proof. By (3.1.1), Corollary 2.5.17 and Point 6 of Theorem 3.4.2,

$$
\begin{array}{l}
\mathsf{P} \approx_\alpha \mathsf{P}' \iff \forall \mathbf{G} \in \mathit{Goals}.\, \alpha(\mathcal{B}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}]\!]) = \alpha(\mathcal{B}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}']\!]) \iff \\
\forall \mathbf{G} \in \mathit{Goals}.\, \alpha(\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}]\!]) = \alpha(\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}']\!]) \iff \\
\forall \mathbf{G} \in \mathit{Goals}.\, \mathcal{Q}_\alpha[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}]\!] = \mathcal{Q}_\alpha[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}']\!].
\end{array}
$$

Now the proof is analogous to the one of Corollary 2.5.6, by using $\mathcal{Q}_\alpha[\![\cdot]\!]$ and $\mathcal{F}_\alpha[\![\cdot]\!]$ instead of $\mathcal{B}_\alpha[\![\cdot]\!]$ and $\mathcal{O}_\alpha[\![\cdot]\!]$, respectively. By definition of $\mathcal{F}_\alpha[\![\cdot]\!]$, the minimality is trivial. The proof of the converse is by contradiction, by using the *AND*-compositionality property of $\mathcal{Q}_\alpha[\![\cdot]\!]$ and by structural induction on the goal $\mathbf{G}$, such that $\mathcal{Q}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!] \neq \mathcal{Q}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}']\!]$. ■

Theorem 3.4.4 *Let α be a denotational observable. Then*

1. $\mathcal{F}_\alpha[\![\mathsf{P}]\!] \leq \mathcal{O}_\alpha[\![\mathsf{P}]\!]$
2. $\mathcal{Q}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!] \leq \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!]$.

Proof. We prove the points separately.

Point 1 First of all observe that, by Point 4 of Theorem 3.4.2, $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!]$ and, by Theorem 2.5.15, $\mathcal{O}[\![\mathsf{P}]\!] = \mathcal{F}[\![\mathsf{P}]\!]$. Then the proof follows by observing that, since $\mathcal{O}_\alpha[\![\mathsf{P}]\!]$ is correct, $\alpha(\mathcal{O}[\![\mathsf{P}]\!]) \leq \mathcal{O}_\alpha[\![\mathsf{P}]\!]$.

Point 2 First of all observe that by Point 6 of Theorem 3.4.2, $\alpha(\mathcal{Q}[\![\mathbf{G}\ in\ \mathsf{P}]\!]) = \mathcal{Q}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!]$ and by Corollary 2.5.17, $\mathcal{Q}[\![\mathbf{G}\ in\ \mathsf{P}]\!] = \mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!]$. Then the proof follows by observing that, since $\mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!]$ is correct, $\alpha(\mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!]) \leq \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!]$. ■

Now we make an assumption to simplify the notation of examples. Consider a goal $\mathsf{A}_1, \ldots, \mathsf{A}_n$, an (abstract) interpretation $\mathsf{X} \in \mathbb{I}_\mathbb{A}$ and an expression involving $\mathsf{X}(\mathsf{A}_1), \ldots, \mathsf{X}(\mathsf{A}_n)$. In the following we assume that, for any occurrence of $\mathsf{X}(\mathsf{A}_i)$, all the variables in $var(\mathsf{X}(\mathsf{A}_i)) \setminus var(\mathsf{A}_i)$ are renamed apart from all the variables in any other $\mathsf{X}(\mathsf{A}_j)$ in the expression. This can always be obtained by choosing a suitable representative of X.

3.4.1 The Computed Answer Observable and the s-semantics

We show now how to reconstruct the s-semantics [45, 13] by means of the observable ξ of Example 3.1.3. We can prove that ξ is indeed a denotational observable. By using a slight simplification of the arguments of Example 3.3.2, it can be proved that the abstract operation $\widetilde{\sum}$ turns out to be point-wise union while $\tilde{\cdot}$ and $\tilde{\times}$ are

$$\begin{aligned}
\mathsf{A} \tilde{\cdot} \mathsf{X} = \phi\left[{}^{\Theta}/_{\mathsf{A}}\right] \text{ where } \Theta := \big\{\vartheta \mid \langle \mathsf{H}, \Theta' \rangle &\text{ is a renamed apart (from} \\
&\mathsf{A}) \text{ version of } \langle \mathsf{A}', \mathsf{X}(\mathsf{A}') \rangle, \text{ for some } \mathsf{A}' \leq \mathsf{A},\ \vartheta' \in \Theta' \\
&\text{and } \vartheta = mgu(\mathsf{A}, \mathsf{H}\vartheta')|_{\mathsf{A}}\big\}, \\
\mathsf{X}_1 \tilde{\times} \mathsf{X}_2 = \lambda \mathbf{G}. \big\{\vartheta \mid \mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2), &\text{ for } i = 1, 2,\ \mathbf{G}'_i \equiv \mathbf{G}_i,\ \vartheta_i \text{ is a renamed} \\
&\text{version of an element in } \mathsf{X}_i(\mathbf{G}'_i), \text{ via a renaming } \rho_i \text{ s.t.} \\
&\mathbf{G}'_i\rho_i = \mathbf{G}_i,\ var(\vartheta_1) \cap var(\vartheta_2) \subseteq var(\mathbf{G}_1) \cap var(\mathbf{G}_2) \\
&\text{and } \vartheta := (\vartheta_1 \circ mgu(\mathbf{G}_2\vartheta_1, \mathbf{G}_2\vartheta_2))|_{\mathbf{G}}\big\}.
\end{aligned}$$

Example 3.4.5

Consider the program P of Figure 3.1 and the collections

$$\mathsf{X}_1 := \begin{cases} \mathsf{mul}(v, y, x) \mapsto \{\{v/0, x/0\}\} \\ \mathsf{sum}(x, y, z) \mapsto \{\{y/a, x/f(z)\}\} \end{cases}$$
$$\mathsf{X}_2 := \mathsf{sum}(x, y, z) \mapsto \{\{x/0, z/y\}, \ldots, \{x/s^n(0), z/s^n(y)\}, \ldots\}$$

By applying the abstract operations we obtain

$$\begin{aligned} \mathsf{mul}(x, a, y) \tilde{\cdot} \mathsf{X}_1 &= \mathsf{mul}(x, a, y) \mapsto \{\{x/0, y/0\}\} \\ \mathsf{sum}(x, y, s^2(0)) \tilde{\cdot} \mathsf{X}_2 &= \mathsf{sum}(x, y, s^2(0)) \mapsto \{\{x/0, y/s^2(0)\}, \{x/s(0), y/s(0)\}, \\ &\qquad \{x/s^2(0), y/0\}\} \\ \mathsf{sum}(x, y, g(y)) \tilde{\cdot} \mathsf{X}_1 &= \mathsf{sum}(x, y, g(y)) \mapsto \{\{x/f(g(a)), y/a\}\} \end{aligned}$$

and

$$\mathsf{X}_1 \tilde{\times} \mathsf{X}_2 = \begin{cases} \mathsf{mul}(v, y, x), \mathsf{sum}(x, y, z) \mapsto \{\{v/0, x/0, z/y\}\} \\ \mathsf{sum}(x, y, z), \mathsf{sum}(x, y, z) \mapsto \emptyset \end{cases}$$

Note that in the collection $\mathsf{X}_1 \tilde{\times} \mathsf{X}_2$ we can not compute any solution for the goal $\mathsf{sum}(x, y, z), \mathsf{sum}(x, y, z)$ since the same variable x should be bound, at the same time, to $f(z)$ and to some $s^n(0)$.

The abstract semantic function is

$$\begin{aligned} \tilde{\mathcal{C}}[\![p(\mathbf{t}) \leftarrow \mathbf{B}]\!]_{\mathsf{X}} = \lambda p(\mathbf{x}). \{\vartheta \mid \mathbf{x} &\text{ are new variables, } \vartheta_i \in \mathsf{X}(p_i(\mathbf{x}_i)) \text{ and} \\ &\vartheta := (\{\mathbf{x}/\mathbf{t}\} \circ \mathit{mgu}(\mathbf{B}, (p_1(\mathbf{x}_1)\vartheta_1, \ldots, p_n(\mathbf{x}_n)\vartheta_n)))|_{\mathbf{x}}\}. \end{aligned}$$

The abstraction of the top-down denotation is

$$\xi(\mathcal{O}[\![P]\!]) = \lambda p(\mathbf{x}). \{\vartheta \mid p(\mathbf{x}) \xrightarrow[P]{\vartheta}{}^* \square\}/_{\equiv_{\mathbb{A}_{ca}}} = \mathcal{F}_\xi[\![P]\!]$$

and is isomorphic to the top-down definition of the s-semantics $\mathcal{O}_{ca}[\![P]\!]$ (see Section 1.3.4). Indeed it is easy to see that it is just a matter of representation. In the s-semantics case, the substitution is simply applied to the pure atom, while in our case, given the pure atom, the corresponding substitution is returned. The same isomorphism holds between the abstract semantic function

$$\begin{aligned} \mathcal{P}_\xi[\![P]\!]_{\mathsf{X}} = \lambda p(\mathbf{x}). \{\vartheta \mid p(\mathbf{t}) \leftarrow B_1, \ldots, B_n \in P, \mathbf{x} &\text{ are new variables,} \\ &\vartheta_i \in \mathsf{X}(p_i(\mathbf{x}_i)) \text{ and } \vartheta = (\{\mathbf{x}/\mathbf{t}\} \circ \mathit{mgu}(\mathbf{B}, \\ &(p_1(\mathbf{x}_1)\vartheta_1, \ldots, p_n(\mathbf{x}_n)\vartheta_n)))|_{\mathbf{x}}\} \end{aligned}$$

```
mul(0,X,0).
mul(s(V),Y,Z) :- mul(V,Y,X),sum(X,Y,Z).

sum(0, X, X).
sum(s(X), Y, s(Z)) :- sum(X,Y,Z).
```

Figure 3.1: The program of Example 3.4.5

and the immediate consequence operator of the s-semantics $\mathsf{T}_{\mathsf{P}}^{\mathsf{ca}}$. From Theorem 3.4.2 we can derive the usual properties of the s-semantics, namely that $\mathcal{F}_{\mathcal{E}}[\![\mathsf{P}]\!]$ is correct and minimal w.r.t. computed answers and that the answers computed for any goal can be obtained from the answers computed for pure atomic goals.

Example 3.4.6

Consider again the program P of Figure 3.1. By repeatedly applying the definition of $\mathcal{P}_{\mathcal{E}}[\![\cdot]\!]$ we obtain

$$\mathcal{P}_{\mathcal{E}}[\![\mathsf{P}]\!]\uparrow_1 = \begin{cases} \mathsf{mul}(x,y,z) \mapsto \{\{x/0, z/0\}\} \\ \mathsf{sum}(x,y,z) \mapsto \{\{x/0, z/y\}\} \end{cases}$$

$$\mathcal{P}_{\mathcal{E}}[\![\mathsf{P}]\!]\uparrow_2 = \begin{cases} \mathsf{mul}(x,y,z) \mapsto \{\{x/0, z/0\}, \{x/s(0), z/y\}\} \\ \mathsf{sum}(x,y,z) \mapsto \{\{x/0, z/y\}, \{x/s(0), z/s(y)\}\} \end{cases}$$

$$\mathcal{P}_{\mathcal{E}}[\![\mathsf{P}]\!]\uparrow_3 = \begin{cases} \mathsf{mul}(x,y,z) \mapsto \{\{x/0, z/0\}, \{x/s(0), z/y\}, \{x/s^2(0), y/0, z/0\}, \\ \qquad\qquad \{x/s^2(0), y/s(0), z/s^2(0)\}\} \\ \mathsf{sum}(x,y,z) \mapsto \{\{x/0, z/y\}, \{x/s(0), z/s(y)\}, \{x/s^2(0), z/s^2(y)\}\} \end{cases}$$

$$\ldots$$

The fixpoint $\mathcal{F}_{\mathcal{E}}[\![\mathsf{P}]\!]$ can be reached only at ω since the program computes infinite (different) solutions.

3.4.2 The Correct Answer Observable

We show how to obtain a semantics which models correct answers and it is isomorphic to the atomic logical consequence semantics (c-semantics) [16, 45]. A similar construction can be used to define ground correct answers leading to the least Herbrand model semantics.

Correct answers are closed under instantiation. This property corresponds to the downward closure of the corresponding substitution sets. These sets can be (efficiently) handled with maximal substitution sets w.r.t. subsumption ($\leq$).

First of all let Θ be a substitution set modulo variance. Θ is *non-redundant* (w.r.t. $\leq$) if and only if $\forall \vartheta, \vartheta' \in \Theta. \vartheta \neq \vartheta' \Longrightarrow \vartheta \not\leq \vartheta'$. We denote by $\lceil \cdot \rceil$ the non-redundancy operator, i.e., $\lceil \Theta \rceil := \{\vartheta \mid \vartheta \in \Theta, \forall \vartheta' \in \Theta. \vartheta \neq \vartheta' \Longrightarrow \vartheta \not\leq \vartheta'\}$. Any downward closed (correct answer) substitution set Θ can be (uniquely) represented by the non-redundant substitution set $\lceil \Theta \rceil$. Furthermore, since any correct answer is an instance of some computed answer, given a computed answer substitution set Θ, $\lceil \Theta \rceil$ is the representative of the corresponding correct answer substitution set.

The abstract domain is $\mathbb{A}_{\mathrm{lc}} \subseteq [\mathit{Goals} \rightharpoonup \wp(\mathit{Subst})]$ and the *correct answer* observable $\delta : \mathbb{C} \to \mathbb{A}_{\mathrm{lc}}$ can be obtained by further abstraction of ξ, i.e.,

$$\delta(\mathsf{D}) := \lambda \mathbf{G}. \lceil \xi(\mathsf{D}(\mathbf{G})) \rceil.$$

By applying the definitions, the abstract operators are $\mathsf{A} \tilde{\cdot} \mathsf{X}_1 = \lambda \mathbf{G}. \lceil (\mathsf{A} \cdot_{\mathrm{ca}} \mathsf{X}_1)(\mathbf{G}) \rceil$, $\mathsf{X}_1 \tilde{\times} \mathsf{X}_2 = \lambda \mathbf{G}. \lceil (\mathsf{X}_1 \times_{\mathrm{ca}} \mathsf{X}_2)(\mathbf{G}) \rceil$, $\mathsf{X}_1 \tilde{+} \mathsf{X}_2 = \lambda \mathbf{G}. \lceil (\mathsf{X}_1 +_{\mathrm{ca}} \mathsf{X}_2)(\mathbf{G}) \rceil$.

δ is a denotational observable and its immediate consequence operator is

$$\mathcal{P}_\delta[\![\mathsf{P}]\!]_\mathsf{X} = \lambda \mathsf{p}(\mathbf{x}). \lceil \mathcal{P}_\xi[\![\mathsf{P}]\!]_\mathsf{X}(\mathsf{p}(\mathbf{x})) \rceil.$$

As was for the s-semantics (see Section 3.4.1), it can be proved that the abstract denotation $\mathcal{F}_\delta[\![\mathsf{P}]\!]$ is isomorphic to $\lceil \mathcal{O}_{\mathrm{lc}}[\![\mathsf{P}]\!] \rceil$, where $\mathcal{O}_{\mathrm{lc}}[\![\mathsf{P}]\!]$ is the atomic logical consequence semantics [16, 47].

3.4.3 The Call Pattern Observable

The call patterns (with state) of a program P for a goal $\mathbf{G}$ are the atoms selected in any *SLD*-derivation of $\mathbf{G}$ in P. A call pattern semantics was defined in [53] and used as a basis for call pattern analysis in [51]. We define the domain $\mathbb{A}_{\mathrm{cp}} \subseteq [\mathit{Goals} \rightharpoonup \wp(\mathit{Atoms} \times \mathit{Subst})]$. For any $\mathsf{X} \in \mathbb{A}_{\mathrm{cp}}$ the interpretation of $\langle \mathsf{C}, \vartheta \rangle \in \mathsf{X}(\mathbf{G})$ is "the execution of the goal $\mathbf{G}$ generates a procedure call C with state (partial computed answer substitution) ϑ". Note that C can be $\square$. The *call pattern* observable η is

$$\begin{aligned}
\eta(\mathsf{D}) &:= \lambda \mathbf{G}. \{\langle \mathsf{C}, \mathit{answer}(\mathsf{d}) \rangle \mid \mathsf{d} \in \mathsf{D}(\mathbf{G}), \mathit{last}(\mathsf{d}) = (\mathsf{C}, \mathbf{B}), \\
&\qquad\qquad \mathbf{B} \neq \square \Longrightarrow \mathsf{C} \neq \square\}. \\
\eta^\gamma(\mathsf{X}) &:= \lambda \mathbf{G}. \mathit{wf}_\mathbf{G}(\{\mathsf{d} \mid \mathit{first}(\mathsf{d}) = \mathbf{G}, \mathit{last}(\mathsf{d}) = (\mathsf{C}, \mathbf{B}), \\
&\qquad\qquad \langle \mathsf{C}, \mathit{answer}(\mathsf{d}) \rangle \in \mathsf{X}(\mathbf{G}), \mathbf{B} \neq \square \Longrightarrow \mathsf{C} \neq \square\}).^{8}
\end{aligned}$$

Example 3.4.7

Consider the collections D_1, D_2 of Example 2.1.1 (on page 39). Then

$$\eta(\mathsf{D}_1) = \begin{cases} \mathsf{p}(\mathsf{x}, \mathsf{y}) \mapsto \{\langle \mathsf{p}(\mathsf{x}, \mathsf{y}), \varepsilon \rangle, \langle \mathsf{q}(\mathsf{x}, \mathsf{y}), \varepsilon \rangle, \langle \square, \{\mathsf{x}/\mathsf{g}(\mathsf{a}), \mathsf{y}/\mathsf{b}\} \rangle\} \\ \mathsf{n}(\mathsf{x}) \quad \mapsto \{\langle \mathsf{n}(\mathsf{x}), \varepsilon \rangle, \langle \mathsf{p}(\mathsf{x}, \mathsf{x}), \varepsilon \rangle\} \end{cases}$$

[8] Recall that $\mathit{wf}_\mathbf{G}(\mathsf{S})$ is the (greatest) well-formed subset (of derivations starting from $\mathbf{G}$ only) of any set of derivations S.

$$\eta(D_2) = \begin{cases} q(x,y) & \mapsto \{\langle q(x,y),\ \varepsilon\rangle, \langle n(f(s),x),\ \varepsilon\rangle, \langle \Box, \{x/a\}\rangle\} \\ q(f(x),z) & \mapsto \{\langle q(f(x),z),\ \varepsilon\rangle, \langle m(g(s),f(x)), \{z/a\}\rangle, \\ & \qquad \langle \Box, \{z/a, x/g(s)\}\rangle\} \end{cases}$$

Consider now the collection $\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ P]\!]$ of Example 2.2.1 (on page 43), where P is the append program and $\mathbf{G} := ap([a],[l],x), ap(x,[h],z)$.

$$\begin{aligned} \eta(\mathcal{Q}[\![\mathbf{G}\ \mathit{in}\ P]\!]) = ap([a],[l],x), ap(x,[h],z) \mapsto \{ & \\ & \langle ap([a],[l],x),\ \varepsilon\rangle, \langle ap([\,],[l],w), \{x/[a|w]\}\rangle, \\ & \langle ap([a,l],[h],z), \{x/[a/l]\}\rangle, \langle ap([l],[h],s), \{x/[a|l], z/[a|s]\}\rangle, \\ & \langle ap([\,],[h],s'), \{x/[a|l], z/[a,l/s']\}\rangle, \\ & \langle \Box, \{x/[a|l], z/[a,l,h]\}\rangle\} \end{aligned}$$

Consider the collection $D_1 \triangleright D_2$ of Example 2.2.5 (on page 49). Then

$$\eta(D_1 \triangleright D_2) = \begin{cases} p(x,y) & \mapsto \{\langle p(x,y),\ \varepsilon\rangle, \langle q(x,y),\ \varepsilon\rangle, \langle n(f(s),x),\ \varepsilon\rangle, \\ & \qquad \langle \Box, \{x/a\}\rangle, \langle \Box, \{x/g(s), y/b\}\rangle\} \\ n(x) & \mapsto \{\langle n(x),\ \varepsilon\rangle, \langle p(x,x),\ \varepsilon\rangle\} \end{cases}$$

Consider the collection D'''' of Example 2.3.1 (on page 54). Then

$$\begin{aligned} \eta(D'''') = anc(x,y) \mapsto \{ & \langle anc(x,y),\ \varepsilon\rangle, \langle par(x,y),\ \varepsilon\rangle, \langle \Box, \{x/joe, y/jay\}\rangle, \\ & \langle \Box, \{x/jay, y/tim\}\rangle, \langle par(x,z),\ \varepsilon\rangle, \langle anc(jay,y), \{x/joe\}\rangle, \\ & \langle anc(tim,y), \{x/jay\}\rangle\} \end{aligned}$$

Note that ξ can be obtained by further abstraction of η, by collecting only the substitutions corresponding to the goal $\Box$, i.e., $\xi(D)(\mathbf{G}) = \{\vartheta \mid \langle \vartheta,\ \Box\rangle \in \eta(D)(\mathbf{G})\}$.

The axioms (3.4.1), (3.4.2), (3.4.3) are satisfied, hence η is a denotational observable. The operation $\widetilde{\sum}$ turns out to be point-wise union while $\tilde{\cdot}$ and $\tilde{\times}$ are:

$$\begin{aligned} A \tilde{\cdot} X = \phi\left[{}^{R}/_{A}\right] \text{ where } R := \{ & \langle C\vartheta,\ \vartheta|_A\rangle \mid \langle H,\ R'\rangle \text{ is a renamed apart} \\ & \text{(from } A\text{) version of } \langle A',\ X(A')\rangle, \text{ for some } A' \leq A, \\ & \langle C,\ \vartheta'\rangle \in R' \text{ and } \vartheta := mgu(A, H\vartheta')\}, \\ X_1 \tilde{\times} X_2 = \lambda\mathbf{G}.\{ & \langle C,\ \vartheta\rangle \mid \mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2), \text{ for } i = 1,2,\ \mathbf{G}'_i \equiv \mathbf{G}_i, \\ & r_i = \langle C_i,\ \vartheta_i\rangle \text{ is a renamed version of an element in } X_i(\mathbf{G}'_i) \\ & \text{via a renaming } \rho_i \text{ s.t. } \mathbf{G}'_i\rho_i = \mathbf{G}_i, \\ & var(r_1) \cap var(r_2) \subseteq var(\mathbf{G}_1) \cap var(\mathbf{G}_2) \text{ and if } C_1 \neq \Box \text{ then} \\ & \langle C,\ \vartheta\rangle := \langle C_1,\ \vartheta_1\rangle \\ & \text{else } \vartheta := (\vartheta_1 \circ mgu(\mathbf{G}_2\vartheta_1, \mathbf{G}_2\vartheta_2))|_G,\ C := C_2\vartheta\}. \end{aligned}$$

Example 3.4.8

Consider the collections.

$$X_1 := \begin{cases} p(x,y) \mapsto \big\{\langle p(x,y),\ \varepsilon\rangle, \langle q(x,y),\ \varepsilon\rangle, \langle \Box,\ \{x/g(a), y/b\}\rangle\big\} \\ n(x) \quad \mapsto \big\{\langle n(x),\ \varepsilon\rangle, \langle p(x,x),\ \varepsilon\rangle\big\} \end{cases}$$

$$X_2 := \begin{cases} q(x,y) \quad \mapsto \big\{\langle q(x,y),\ \varepsilon\rangle, \langle n(f(s),x),\ \varepsilon\rangle, \langle \Box,\ \{x/a\}\rangle\big\} \\ q(f(x),z) \mapsto \big\{\langle q(f(x),z),\ \varepsilon\rangle, \langle m(g(s),f(x)),\ \{z/a\}\rangle, \\ \qquad\qquad \langle \Box,\ \{z/a, x/g(s)\}\rangle\big\} \end{cases}$$

Then

$$X_1 \mathbin{\widetilde{\times}} X_2 = \begin{cases} p(x,y), q(x,y) \quad \mapsto \big\{\langle p(x,y),\ \varepsilon\rangle, \langle q(x,y),\ \varepsilon\rangle, \\ \qquad\qquad \langle n(f(s),g(a)),\ \{x/g(a), y/b\}\rangle\big\} \\ p(x,y), q(f(x),z) \mapsto \big\{\langle p(x,y),\ \varepsilon\rangle, \langle q(x,y),\ \varepsilon\rangle, \\ \qquad\qquad \langle m(g(s),f(g(a))),\ \{x/g(a), y/b, z/a\}\rangle, \\ \qquad\qquad \langle \Box,\ \{x/g(a), y/b, z/a\}\rangle\big\} \\ n(x), q(x,y) \quad \mapsto \big\{\langle n(x),\ \varepsilon\rangle, \langle p(x,x),\ \varepsilon\rangle\big\} \\ n(x), q(f(x),z) \quad \mapsto \big\{\langle n(x),\ \varepsilon\rangle, \langle p(x,x),\ \varepsilon\rangle\big\} \end{cases}$$

$$p(a,x) \mathbin{\widetilde{\cdot}} X_1 = p(a,x) \mapsto \big\{\langle p(a,x),\ \varepsilon\rangle, \langle q(a,x),\ \varepsilon\rangle\big\}$$

and

$$\begin{aligned} p(g(x),z) \mathbin{\widetilde{\cdot}} X_1 = p(g(x),z) &\mapsto \big\{\langle p(g(x),z),\ \varepsilon\rangle, \langle q(g(x),z),\ \varepsilon\rangle, \\ &\qquad \langle \Box,\ \{x/a, z/b\}\rangle\big\} \\ q(f(a),x) \mathbin{\widetilde{\cdot}} X_2 = q(f(a),x) &\mapsto \big\{\langle q(f(a),x),\ \varepsilon\rangle, \langle n(f(s),f(a)),\ \varepsilon\rangle, \\ &\qquad \langle m(g(s),f(a)),\ \{x/a\}\rangle\big\} \end{aligned}$$

Maybe it is useful to recall that the $\widetilde{\cdot}$ operation gives a collection which is defined only on its first argument atom.

The abstract semantic function is

$$\begin{aligned} \widetilde{\mathcal{C}}[\![p(\mathbf{t}) \leftarrow \mathbf{B}]\!]_X = \lambda p(\mathbf{x}).\big\{\langle C\vartheta',\ \vartheta\rangle \mid \mathbf{B} = (\mathbf{B}', \mathbf{B}''),\ &\mathbf{x} \text{ are new variables,} \\ &\exists k \text{ s.t. } \forall i < k.\ \langle \Box,\ \vartheta_i\rangle \in X(p_i(\mathbf{x}_i)),\ \langle C,\ \vartheta_k\rangle \in X(p_k(\mathbf{x}_k)), \\ &\vartheta := (\{\mathbf{x}/\mathbf{t}\} \circ \vartheta')\,|_{\mathbf{x}},\ \vartheta' := \mathit{mgu}\,\big(\mathbf{B}', (p_1(\mathbf{x}_1)\vartheta_1, \ldots, p_k(\mathbf{x}_k)\vartheta_k)\big) \\ &\text{and } \mathbf{B}'' \neq \Box \Longrightarrow C \neq \Box\big\}. \end{aligned}$$

The proof that $\eta : \mathbb{C} \to \mathbb{A}_{cp}$ is an observable and that the operations are those defined is analogous to that one given for the previous observables and therefore it is omitted.

3.5 Semi-denotational Observables

Semi-denotational observables are intended to model some of the properties useful for static program analysis, where approximation plays a major role and we are forced to give up precision (to get effectivity). Therefore, we relax the optimality conditions of denotational observables axioms to admit non-precise $\widetilde{\times}$ and $\widetilde{\triangleright}$ operators. However, we guarantee that weak[9] (compositionality) properties are still satisfied. Hence every denotational observable is a semi-denotational observable, but the converse does not hold.

Definition 3.5.1 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be an observable. Then α is a* semi-denotational *observable if, for any $\mathsf{A} \in$ Atoms, $\mathsf{D}', \mathsf{D}'' \in \mathbb{C}$, $\mathsf{D} \in \mathbb{P}\mathbb{C}$, $\mathbf{G} \in$ Goals and chain $\{\mathsf{D}_j\}_{j\in J} \subseteq \mathbb{P}\mathbb{C}$ the following properties hold.*

$$\widetilde{\cdot} \text{ and } \widetilde{\times} \text{ distribute over } \widetilde{\sum}, \tag{3.5.1}$$

$$\alpha(\mathsf{A} \cdot \gamma\alpha(Id_{\mathbb{I}})) = \alpha(\mathsf{A} \cdot Id_{\mathbb{I}}) = \alpha(\phi_{\mathsf{A}}), \tag{3.5.2}$$

$$\alpha(\gamma\alpha(\mathsf{D}') \triangleright \gamma\alpha(Id_{\mathbb{C}})) = \alpha(\mathsf{D}' \triangleright Id_{\mathbb{C}}) = \alpha(\mathsf{D}'), \tag{3.5.3}$$

$$\alpha(\gamma\alpha(\mathsf{D}') \times \gamma\alpha(\phi_{\mathbf{G}})) = \alpha(\mathsf{D}' \times \phi_{\mathbf{G}}), \tag{3.5.4}$$

$$\alpha(\gamma\alpha(\mathsf{D}') \triangleright \gamma\alpha(su(\gamma\alpha(\mathsf{D})))) = \alpha(\gamma\alpha(\mathsf{D}') \triangleright su(\gamma\alpha(\mathsf{D}))), \tag{3.5.5}$$

$$\alpha(\mathsf{D} \triangleright pu(\gamma\alpha(\sum\{\mathsf{D}_j\}_{j\in J}))) = \alpha(\mathsf{D} \triangleright pu(\sum\{\gamma\alpha(\mathsf{D}_j)\}_{j\in J})), \tag{3.5.6}$$

$$\begin{aligned}&\alpha(\gamma\alpha(\mathsf{D}') \times \gamma\alpha(\gamma\alpha(\mathsf{D}'') \triangleright su(\gamma\alpha(\mathsf{D})))) = \\ &\qquad \alpha(\gamma\alpha(\mathsf{D}') \times (\gamma\alpha(\mathsf{D}'') \triangleright su(\gamma\alpha(\mathsf{D})))),\end{aligned} \tag{3.5.7}$$

$$\alpha(\mathsf{A} \cdot \gamma\alpha(\gamma\alpha(\mathsf{D}') \triangleright su(\gamma\alpha(\mathsf{D})))) = \alpha(\mathsf{A} \cdot (\gamma\alpha(\mathsf{D}') \triangleright su(\gamma\alpha(\mathsf{D})))), \tag{3.5.8}$$

$$\begin{aligned}&\alpha(\gamma\alpha(\mathsf{D}') \triangleright \gamma\alpha(\gamma\alpha(\mathsf{D}'') \triangleright su(\gamma\alpha(\mathsf{D})))) = \\ &\qquad \alpha(\gamma\alpha(\mathsf{D}') \triangleright (\gamma\alpha(\mathsf{D}'') \triangleright su(\gamma\alpha(\mathsf{D})))).\end{aligned} \tag{3.5.9}$$

The following theorem shows that under these conditions (as in the denotational case) we can just replace $\mathcal{C}_\alpha[\![\cdot]\!]$ by the optimal abstract version $\widetilde{\mathcal{C}}[\![\cdot]\!]$ of $\mathcal{C}[\![\cdot]\!]$ (see (3.4.4)) to make the definition precise.

Theorem 3.5.2 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-denotational observable and $\mathsf{X} \in \mathbb{P}\mathbb{A}$. Then*

1. $\mathcal{P}_\alpha[\![\mathsf{P}]\!]_{\mathsf{X}} = \alpha(\mathcal{P}[\![\mathsf{P}]\!]_{\gamma(\mathsf{X})})$
2. *$\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ is continuous on $\mathbb{A}$ and $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathcal{P}_\alpha[\![\mathsf{P}]\!]\uparrow\omega$.*

[9] By weak properties we mean that we cannot ensure the precise properties of Sections 3.3 and 3.4, but their approximated formulation, where we replace the equality symbol by inequality. For example, $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!]$ becomes $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) \leq \mathcal{F}_\alpha[\![\mathsf{P}]\!]$.

Proof. We prove the points separately.

Point 1 The proof is straightforward by definition of $\widetilde{\mathcal{C}}[\![\cdot]\!]$, by (3.3.1) and by definition of $\mathcal{P}_\alpha[\![\cdot]\!]$ and hence is omitted.

Point 2 Let $\{\mathsf{X}_i\}_{i\in I} \subseteq \mathbb{PA}$ be a chain. Since $\widetilde{\sum}$ is the *lub* operation on $\mathbb{PA}$, we have to prove that $\widetilde{\sum}\{\mathcal{P}_\alpha[\![\mathsf{P}]\!]_{\mathsf{X}_i}\}_{i\in I} = \mathcal{P}_\alpha[\![\mathsf{P}]\!]_{\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I}}$. The following equalities hold.

$$
\begin{aligned}
&\widetilde{\sum}\{\mathcal{P}_\alpha[\![\mathsf{P}]\!]_{\mathsf{X}_i}\}_{i\in I} = \\
&\qquad [\text{by definition of } \widetilde{\sum} \text{ and by Point 1}] \\
&\alpha(\sum\{\gamma\alpha(\mathcal{P}[\![\mathsf{P}]\!]_{\gamma(\mathsf{X}_i)})\}_{i\in I}) = \\
&\qquad [\text{by (3.3.1)}] \\
&\alpha(\sum\{\mathcal{P}[\![\mathsf{P}]\!]_{\gamma(\mathsf{X}_i)}\}_{i\in I}) = \\
&\qquad [\text{since } \mathcal{P}[\![\mathsf{P}]\!] \text{ is continuous}] \\
&\alpha(\mathcal{P}[\![\mathsf{P}]\!]_{\sum\{\gamma(\mathsf{X}_i)\}_{i\in I}}) = \\
&\qquad [\text{by (2.5.8) and (3.3.1)}] \\
&\alpha(Id_{\mathbb{I}} + \gamma\alpha(tree(\mathsf{P}) \triangleright pu(\sum\{\gamma(\mathsf{X}_i)\}_{i\in I}))) = \\
&\qquad [\text{since } \alpha\gamma = Id \text{ and by (3.5.6)}] \\
&\alpha(Id_{\mathbb{I}} + \gamma\alpha(tree(\mathsf{P}) \triangleright pu(\gamma\alpha(\sum\{\gamma(\mathsf{X}_i)\}_{i\in I}))) = \\
&\qquad [\text{by (3.3.1) and (2.5.8)}] \\
&\alpha(\mathcal{P}[\![\mathsf{P}]\!]_{\gamma\alpha(\sum\{\gamma(\mathsf{X}_i)\}_{i\in I})}) = \\
&\qquad [\text{by Point 1}] \\
&\mathcal{P}_\alpha[\![\mathsf{P}]\!]_{\alpha(\sum\{\gamma(\mathsf{X}_i)\}_{i\in I})}) = \\
&\qquad [\text{by definition of } \widetilde{\sum}] \\
&\mathcal{P}_\alpha[\![\mathsf{P}]\!]_{\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I}}.
\end{aligned}
$$

■

Lemma 3.5.3 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-denotational observable, $\mathsf{X} \in \mathbb{PA}$ and $\mathsf{D}' \in \mathbb{C}$.*

1. $\alpha(\gamma\alpha(\mathsf{D}') \times \gamma\alpha(Id_{\mathbb{C}})) = \alpha(\mathsf{D}' \times Id_{\mathbb{C}})$,
2. $su_\alpha(\mathsf{X}) = \alpha(su(\gamma(\mathsf{X})))$,
3. $\alpha(su(\gamma\alpha(Id_{\mathbb{I}}))) \leq \alpha(Id_{\mathbb{I}})$.

Proof. We prove the points separately.

Point 1

$$\begin{aligned}
&\alpha(\gamma\alpha(\mathsf{D}') \times \gamma\alpha(Id_{\mathbb{C}})) = && [\text{by definition of } Id_{\mathbb{C}} \text{ and } (3.3.1)]\\
&\alpha(\mathsf{D}') \mathbin{\widetilde{\times}} \widetilde{\sum}\{\alpha(\phi_{\mathsf{G}})\}_{\mathsf{G}\in Goals} = && [\text{by } (3.5.1)]\\
&\widetilde{\sum}\{\alpha(\mathsf{D}') \mathbin{\widetilde{\times}} \alpha(\phi_{\mathsf{G}})\}_{\mathsf{G}\in Goals} = && [\text{by definition of } \widetilde{\times} \text{ and by } (3.5.4)]\\
&\widetilde{\sum}\{\alpha(\mathsf{D}' \times \phi_{\mathsf{G}})\}_{\mathsf{G}\in Goals} = && [\text{by definition of } \widetilde{\sum} \text{ and by } (3.3.1)]\\
&\alpha(\sum\{\mathsf{D}' \times \phi_{\mathsf{G}}\}_{\mathsf{G}\in Goals}) = && [\text{by Lemma } 2.2.7]\\
&\alpha(\mathsf{D}' \times \sum\{\phi_{\mathsf{G}}\}_{\mathsf{G}\in Goals}) = && [\text{by definition of } Id_{\mathbb{C}}]\\
&\alpha(\mathsf{D}' \times Id_{\mathbb{C}}).
\end{aligned}$$

Point 2

$$\begin{aligned}
&su_{\alpha}(\mathsf{X}) =\\
&\qquad [\text{by definition of } su_{\alpha}]\\
&\widetilde{\sum}\{(\mathsf{A} \mathbin{\widetilde{\cdot}} \mathsf{X}) \mathbin{\widetilde{\times}} \alpha(Id_{\mathbb{C}})\}_{\mathsf{A}\in Atoms} =\\
&\qquad [\text{by definition of } \widetilde{\sum}, \mathbin{\widetilde{\cdot}} \text{ and } \widetilde{\times}]\\
&\alpha(\sum\{\gamma\alpha(\gamma\alpha(\mathsf{A}\cdot\gamma(\mathsf{X})) \times \gamma\alpha(Id_{\mathbb{C}}))\}_{\mathsf{A}\in Atoms}) =\\
&\qquad [\text{by Point } 1]\\
&\alpha(\sum\{\gamma\alpha((\mathsf{A}\cdot\gamma(\mathsf{X})) \times Id_{\mathbb{C}})\}_{\mathsf{A}\in Atoms}) =\\
&\qquad [\text{by } (3.3.1)]\\
&\alpha(\sum\{(\mathsf{A}\cdot\gamma(\mathsf{X})) \times Id_{\mathbb{C}}\}_{\mathsf{A}\in Atoms}) =\\
&\qquad [\text{by definition of } su]\\
&\alpha(su(\gamma(\mathsf{X}))).
\end{aligned}$$

Point 3

$$\begin{aligned}
&\alpha(su(\gamma\alpha(Id_{\mathbb{I}}))) =\\
&\qquad [\text{by definition of } su \text{ and by } (3.3.1)]\\
&\alpha(\sum\{\gamma\alpha((\mathsf{A}\cdot\gamma\alpha(Id_{\mathbb{I}})) \times Id_{\mathbb{C}})\}_{\mathsf{A}\in Atoms}) =\\
&\qquad [\text{by Point } 1 \text{ and } (3.5.2)]\\
&\alpha(\sum\{\gamma\alpha(\gamma\alpha(\mathsf{A}\cdot Id_{\mathbb{I}}) \times \gamma\alpha(Id_{\mathbb{C}}))\}_{\mathsf{A}\in Atoms}) =\\
&\qquad [\text{by Point } 1 \text{ and } (3.3.1)]
\end{aligned}$$

$$\alpha(\sum\{(\mathsf{A}\cdot Id_{\mathbb{I}})\times Id_{\mathbb{C}}\}_{\mathsf{A}\in Atoms}) =$$
$$[\text{by definition of } su]$$
$$\alpha(su(Id_{\mathbb{I}})) \leq$$
$$[\text{by definition of } su \text{ and } Id_{\mathbb{C}}]$$
$$\alpha(Id_{\mathbb{C}}).$$

■

Lemma 3.5.4 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-denotational observable, $\mathsf{X} \in \mathbb{A}$ and $\mathsf{k}, \mathsf{n} \geq 0$.*

1. $\alpha(su(\gamma\, unf^{\mathsf{k}}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}})))) \leq unf^{\mathsf{k}}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{C}}))$,
2. $\mathsf{X} \widetilde{\rhd}\, unf^{\mathsf{k}}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{C}})) \leq unf^{\mathsf{k}}_{\mathsf{P},\alpha}(\mathsf{X})$,
3. $su_{\mathsf{n}}(\gamma\, unf^{\mathsf{k}}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))) \sqsubseteq \gamma(unf^{\mathsf{k}\mathsf{n}}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{C}})))$.

Proof. We prove the points separately.

Point 1 The proof is by induction on k. For $\mathsf{k} = 0$ the proof is immediate by definition of $unf^{0}_{\mathsf{P},\alpha}$ and by Point 3 of Lemma 3.5.3. For $\mathsf{n} > 0$ the following facts hold.

$$\alpha(su(\gamma\, unf^{\mathsf{k}}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}})))) =$$
$$[\text{by definition of } unf^{\mathsf{k}}_{\mathsf{P},\alpha}]$$
$$\alpha(su(\gamma(unf^{\mathsf{k}-1}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))\,\widetilde{\rhd}\, su_{\alpha}(\alpha(tree(\mathsf{P})))))) =$$
$$[\text{by definition of } su]$$
$$\alpha(\sum\{(\mathsf{A}\cdot\gamma(unf^{\mathsf{k}-1}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))\,\widetilde{\rhd}\, su_{\alpha}(\alpha(tree(\mathsf{P}))))) \times Id_{\mathbb{C}}\}_{\mathsf{A}\in Atoms}) =$$
$$[\text{by definition of } \widetilde{\rhd}, \text{ by Point 2 of Lemma 3.5.3 and by (3.5.5)}]$$
$$\alpha(\sum\{(\mathsf{A}\cdot\gamma\alpha(\gamma\, unf^{\mathsf{k}-1}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))\rhd su(\gamma\alpha(tree(\mathsf{P}))))) \times Id_{\mathbb{C}}\}_{\mathsf{A}\in Atoms}) =$$
$$[\text{by Point 1 of Lemma 3.5.3 and (3.3.1)}]$$
$$\alpha(\sum\{\gamma\alpha(\gamma\alpha(\mathsf{A}\cdot\gamma\alpha(\gamma\, unf^{\mathsf{k}-1}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))\rhd$$
$$su(\gamma\alpha(tree(\mathsf{P}))))) \times Id_{\mathbb{C}})\}_{\mathsf{A}\in Atoms}) =$$
$$[\text{by (3.5.8)}]$$
$$\alpha(\sum\{\gamma\alpha(\gamma\alpha(\mathsf{A}\cdot(\gamma\, unf^{\mathsf{k}-1}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))\rhd su(\gamma\alpha(tree(\mathsf{P}))))) \times Id_{\mathbb{C}})\}_{\mathsf{A}\in Atoms}) =$$
$$[\text{by (3.3.1), Point 1 of Lemma 3.5.3 and by definition of } su]$$
$$\alpha(su(\gamma(unf^{\mathsf{k}-1}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}})))\rhd su(\gamma\alpha(tree(\mathsf{P})))))) \leq$$
$$[\text{by Lemma 2.5.7 and since } \alpha \text{ is monotonic}]$$

$\alpha(su(\gamma(unf^{k-1}_{P,\alpha}(\alpha(Id_{\mathbb{I}})))) \triangleright su(\gamma\alpha(tree(P)))) \leq$

[since $\gamma\alpha$ is extensive and by inductive hypothesis]

$unf^{k-1}_{P,\alpha}(\alpha(Id_{\mathbb{C}})) \widetilde{\triangleright} \alpha(su(\gamma\alpha(tree(P)))) \leq$

[by Point 2 of Lemma 3.5.3 and definition of $unf^{k}_{P,\alpha}$]

$unf^{k}_{P,\alpha}(\alpha(Id_{\mathbb{C}}))$.

Point 2 The proof is by induction on k. For $k = 0$ the proof is immediate by definition of $unf^{0}_{P,\alpha}$ and by (3.5.3). For $n > 0$ the following facts hold.

$X \widetilde{\triangleright} unf^{k}_{P,\alpha}(\alpha(Id_{\mathbb{C}})) =$

[by definition of $unf^{k}_{P,\alpha}$]

$X \widetilde{\triangleright} (unf^{k-1}_{P,\alpha}(\alpha(Id_{\mathbb{C}})) \widetilde{\triangleright} su_{\alpha}(\alpha(tree(P)))) =$

[by definition of $\widetilde{\triangleright}$ and by Point 2 of Lemma 3.5.3]

$\alpha(\gamma(X) \triangleright \gamma\alpha(\gamma(unf^{k-1}_{P,\alpha}(\alpha(Id_{\mathbb{C}}))) \triangleright \gamma\alpha(su(\gamma\alpha(tree(P)))))) =$

[by (3.5.5) and (3.5.9)]

$\alpha(\gamma(X) \triangleright (\gamma(unf^{k-1}_{P,\alpha}(\alpha(Id_{\mathbb{C}}))) \triangleright su(\gamma\alpha(tree(P))))) \leq$

[by the same argument of Lemma 2.2.6 proof and since α is monotonic]

$\alpha((\gamma(X) \triangleright \gamma(unf^{k-1}_{P,\alpha}(\alpha(Id_{\mathbb{I}})))) \triangleright su(\gamma\alpha(tree(P)))) \leq$

[by inductive hypothesis, by definition of $\widetilde{\triangleright}$ and since $\gamma\alpha$ is extensive]

$\alpha(\gamma(unf^{k-1}_{P,\alpha}(X)) \triangleright su(\gamma\alpha(tree(P)))) \leq$

[by definition of $\widetilde{\triangleright}$ and by Point 2 of Lemma 3.5.3]

$unf^{k-1}_{P,\alpha}(X) \widetilde{\triangleright} su_{\alpha}(\alpha(tree(P))) =$

[by definition of $unf^{k}_{P,\alpha}$]

$unf^{k}_{P,\alpha}(X)$.

Point 3 The proof is by induction on n. For $n = 0$ the proof is immediate by definition of su_0. For $n > 0$ the following facts hold.

$su_n(\gamma\, unf^{k}_{P,\alpha}(\alpha(Id_{\mathbb{I}}))) =$

[by definition of su_n]

$su_{n-1}(\gamma\, unf^{k}_{P,\alpha}(\alpha(Id_{\mathbb{I}}))) \triangleright su(\gamma\, unf^{k}_{P,\alpha}(\alpha(Id_{\mathbb{I}}))) \sqsubseteq$

[by inductive hypothesis and since $\gamma\alpha$ is extensive]

$\gamma(unf^{k(n-1)}_{P,\alpha}(\alpha(Id_{\mathbb{C}}))) \triangleright \gamma\alpha(su(\gamma\, unf^{k}_{P,\alpha}(\alpha(Id_{\mathbb{I}})))) \sqsubseteq$

[by Point 1 and since γ is monotonic]

$\gamma(unf^{k(n-1)}_{P,\alpha}(\alpha(Id_{\mathbb{C}}))) \triangleright \gamma(unf^{k}_{P,\alpha}(\alpha(Id_{\mathbb{C}}))) \sqsubseteq$

[since $\gamma\alpha$ is extensive and by definition of $\widetilde{\triangleright}$]

$$\gamma(unf^{k(n-1)}_{P,\alpha}(\alpha(Id_{\mathbb{C}}))\,\widetilde{\triangleright}\,unf^{k}_{P,\alpha}(\alpha(Id_{\mathbb{C}}))) \sqsubseteq$$
$$[\text{by Point 2 and since } \gamma \text{ is monotonic}]$$
$$\gamma(unf^{k}_{P,\alpha}(unf^{k(n-1)}_{P,\alpha}(\alpha(Id_{\mathbb{C}})))) =$$
$$[\text{by definition of } unf^{k}_{P,\alpha}]$$
$$\gamma(unf^{kn}_{P,\alpha}(\alpha(Id_{\mathbb{C}}))).$$

■

The abstract denotational semantics and bottom-up denotation are more precise than the operational counterparts.

Lemma 3.5.5 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-denotational observable. Then*

$$\alpha(\mathcal{O}[\![P]\!]) \leq \mathcal{F}_{\alpha}[\![P]\!] \leq \mathcal{O}_{\alpha}[\![P]\!].$$

Proof. The proof of the first inequality is straightforward by correctness of $\mathcal{F}_{\alpha}[\![P]\!]$ and by Theorem 2.5.15. For the second inequality we prove, by induction on n, that $\forall n \geq 0.\ \mathcal{P}_{\alpha}[\![P]\!]{\uparrow}_{n} \leq \mathcal{O}_{\alpha}[\![P]\!]$. Then the thesis follows by continuity of $\mathcal{P}_{\alpha}[\![\cdot]\!]$ and by definition of $\mathcal{F}_{\alpha}[\![\cdot]\!]$.

First of all observe that since $\widetilde{\triangleright}$ is monotonic and $\widetilde{\sum}$ is the *lub* operation on $\mathbb{A}$, for any $X_i, X \in \mathbb{A}$, $\widetilde{\sum}\{X_i \widetilde{\triangleright} X\}_{i\in I} \leq \widetilde{\sum}\{X_i\}_{i\in I} \widetilde{\triangleright} X$. Then, by a straightforward inductive argument and by (3.2.18),

$$\mathcal{O}_{\alpha}[\![P]\!] \leq \left[\widetilde{\sum}\{unf^{k}_{P,\alpha}(\alpha(Id_{\mathbb{I}}))\}_{k\geq 0}\right] /_{\equiv_{\mathbb{A}}}. \tag{1}$$

Now, we can prove that $\forall n \geq 0.\ \mathcal{P}_{\alpha}[\![P]\!]{\uparrow}_{n} \leq \mathcal{O}_{\alpha}[\![P]\!]$.

$\underline{n = 0|}$ Straightforward, since by definition of $\cdot{\uparrow}_{0}$, $\mathcal{P}_{\alpha}[\![P]\!]{\uparrow}_{0} = \bot_{\mathbb{A}}$.

$\underline{n > 0|}$ The following facts hold.

$$\mathcal{P}_{\alpha}[\![P]\!]{\uparrow}_{n} =$$
$$[\text{by definition of } \cdot{\uparrow}_{n}]$$
$$\mathcal{P}_{\alpha}[\![P]\!]_{\mathcal{P}_{\alpha}[\![P]\!]{\uparrow}_{n-1}} =$$
$$[\text{by Point 1 of Theorem 3.5.2 and by (2.5.8)}]$$
$$\alpha\,(Id_{\mathbb{I}} + tree(P) \triangleright pu(\gamma(\mathcal{P}_{\alpha}[\![P]\!]{\uparrow}_{n-1}))) \leq$$
$$[\text{by inductive hypothesis and by (1)}]$$
$$\alpha\left(Id_{\mathbb{I}} + tree(P) \triangleright pu(\gamma\,\widetilde{\sum}\{unf^{k}_{P,\alpha}(\alpha(Id_{\mathbb{I}}))\}_{k\geq 0})\right) =$$
$$[\text{by definition of } \widetilde{\sum} \text{ and by (3.5.6)}]$$

$$\alpha\left(Id_{\mathbb{I}} + tree(\mathsf{P}) \triangleright pu(\sum\{\gamma(unf^{k}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}})))\}_{k\geq 0})\right) =$$

[by Lemma 2.2.7]

$$\alpha\left(\sum\{Id_{\mathbb{I}} + tree(\mathsf{P}) \triangleright pu(\gamma(unf^{k}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))))\}_{k\geq 0}\right) =$$

[by definition of $\triangleright$ and $Id_{\mathbb{I}}$ and since $\triangleright$ is extensive]

$$\alpha\left(\sum\{(Id_{\mathbb{I}} \triangleright su(tree(\mathsf{P}))) \triangleright pu(\gamma(unf^{k}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))))\}_{k\geq 0}\right) \leq$$

[by Point 2 of Lemma 2.5.12]

$$\alpha\left(\sum\left\{(Id_{\mathbb{I}} \triangleright su(tree(\mathsf{P}))) \triangleright \left(Id_{\mathbb{C}} + \sum\{su_{n}(\gamma(unf^{k}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))))\}_{n\geq 0}\right)\right\}_{k\geq 0}\right) =$$

[by definition of $\triangleright$ and Lemma 2.2.7]

$$\alpha\left(\sum\{(Id_{\mathbb{I}} \triangleright su(tree(\mathsf{P}))) \triangleright su_{n}(\gamma(unf^{k}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))))\}_{n,k\geq 0}\right) \leq$$

[by Point 3 of Lemma 3.5.4 and since α and $\triangleright$ are monotonic]

$$\alpha\left(\sum\{(Id_{\mathbb{I}} \triangleright su(tree(\mathsf{P}))) \triangleright \gamma(unf^{kn}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{C}})))\}_{k,n\geq 0}\right) \leq$$

[by definition of $Id_{\mathbb{I}}$, Lemma 2.2.7 and set-theoretic properties]

$$\alpha\left(\sum\{(\phi_{p(\mathbf{x})} \triangleright su(tree(\mathsf{P}))) \triangleright \gamma(unf^{k}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{C}})))\}_{k\geq 0, p(\mathbf{x})\in Goals}\right) \leq$$

[by definition of $\widetilde{\triangleright}$ and since $\gamma\alpha$ is extensive]

$$\alpha\left(\sum\{\gamma((\alpha(\phi_{p(\mathbf{x})}) \widetilde{\triangleright} su_{\alpha}(\alpha(tree(\mathsf{P})))) \widetilde{\triangleright} unf^{k}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{C}})))\}_{k\geq 0, p(\mathbf{x})\in Goals}\right) \leq$$

[by Point 2 of Lemma 3.5.4]

$$\alpha\left(\sum\{\gamma\left(unf^{k}_{\mathsf{P},\alpha}(\alpha(\phi_{p(\mathbf{x})}) \widetilde{\triangleright} su_{\alpha}(\alpha(tree(\mathsf{P}))))\right)\}_{k\geq 0, p(\mathbf{x})\in Goals}\right) \leq$$

[by definition of $unf^{k}_{\mathsf{P},\alpha}$ and since $\gamma\alpha$ is extensive]

$$\alpha\left(\sum\{\gamma\alpha\left(\sum\{\gamma(unf^{k}_{\mathsf{P},\alpha}(\alpha(\phi_{p(\mathbf{x})})))\}_{k\geq 0}\right)\}_{p(\mathbf{x})\in Goals}\right) =$$

[by definition of $\widetilde{\sum}$]

$$\widetilde{\sum}\left\{\left[\widetilde{\sum}\{unf^{k}_{\mathsf{P},\alpha}(\alpha(\phi_{p(\mathbf{x})}))\}_{k\geq 0}\right]/_{\equiv_{\mathbb{A}}}\right\}_{p(\mathbf{x})\in Goals} =$$

[by (3.2.18)]

$$\mathcal{O}_{\alpha}[\![\mathsf{P}]\!]$$

■

Proposition 3.5.6 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-denotational observable, $\mathsf{X} \in \mathbb{PA}$, $\mathsf{X}', \mathsf{X}'' \in \mathbb{A}$, $\mathsf{A} \in$ Atoms and $\mathbf{G} \in$ Goals. Then the following properties hold.*

1. $\mathsf{A} \tilde{\cdot} (\mathsf{X}' \tilde{\rhd} su_\alpha(\mathsf{X})) \leq (\mathsf{A} \tilde{\cdot} \mathsf{X}') \tilde{\rhd} su_\alpha(\mathsf{X})$,

2. $(\mathsf{X}' \tilde{\rhd} su_\alpha(\mathsf{X})) \tilde{\times} \alpha(\phi_{\mathsf{G}}) \leq (\mathsf{X}' \tilde{\times} \alpha(\phi_{\mathsf{G}})) \tilde{\rhd} su_\alpha(\mathsf{X})$,

3. $\mathsf{X}' \tilde{\times} (\mathsf{X}'' \tilde{\rhd} su_\alpha(\mathsf{X})) \leq (\mathsf{X}' \tilde{\times} \mathsf{X}'') \tilde{\rhd} su_\alpha(\mathsf{X})$.

Proof. We prove only Point 3, essentially using Point 3 of Lemma 2.5.1 and (3.5.7). The proof of the other points is analogous, by using Points 1 and 2 of the above mentioned lemma and (3.5.8) and (3.5.4).

First of all note that in the proof of Point 3 of Lemma 2.5.1 (on page 77) the hypothesis $\mathsf{D}' \rhd su(\mathsf{D}) = \mathsf{D}'$ is not used to prove the inclusion $\mathsf{D}' \times (\mathsf{D}'' \rhd su(\mathsf{D})) \sqsubseteq (\mathsf{D}' \times \mathsf{D}'') \rhd su(\mathsf{D})$ and hence we can conclude that

$$\gamma(\mathsf{X}') \times (\gamma(\mathsf{X}'') \rhd su(\gamma(\mathsf{X}))) \sqsubseteq (\gamma(\mathsf{X}') \times \gamma(\mathsf{X}'')) \rhd su(\gamma(\mathsf{X})). \tag{1}$$

Moreover, by Point 2 of Lemma 3.5.3, by definition of $\tilde{\rhd}$ and $\tilde{\times}$, by (3.5.7) and (3.5.5),

$$su_\alpha(\mathsf{X}) = \alpha(su(\gamma(\mathsf{X}))) \text{ and} \tag{2}$$
$$\mathsf{X}' \tilde{\times} (\mathsf{X}'' \tilde{\rhd} su_\alpha(\mathsf{X})) = \alpha(\gamma(\mathsf{X}') \times (\gamma(\mathsf{X}'') \rhd su(\gamma(\mathsf{X})))). \tag{3}$$

Then the following facts hold.

$$\begin{array}{ll}
\mathsf{X}' \tilde{\times} (\mathsf{X}'' \tilde{\rhd} su_\alpha(\mathsf{X})) = & [\text{by } (3)] \\
\alpha(\gamma(\mathsf{X}') \times (\gamma(\mathsf{X}'') \rhd su(\gamma(\mathsf{X})))) \leq & [\text{by } (1) \text{ and since } \alpha \text{ is monotonic}] \\
\alpha((\gamma(\mathsf{X}') \times \gamma(\mathsf{X}'')) \rhd su(\gamma(\mathsf{X}))) \leq & [\text{since } \gamma\alpha \text{ is extensive}] \\
\alpha(\gamma\alpha(\gamma(\mathsf{X}') \times \gamma(\mathsf{X}'')) \rhd \gamma\alpha(su(\gamma(\mathsf{X})))) = & [\text{by } (2)] \\
\alpha(\gamma\alpha(\gamma(\mathsf{X}') \times \gamma(\mathsf{X}'')) \rhd \gamma(su_\alpha(\mathsf{X}))) = & [\text{by definition of } \tilde{\times} \text{ and } \tilde{\rhd}] \\
(\mathsf{X}' \tilde{\times} \mathsf{X}'') \tilde{\rhd} su_\alpha(\mathsf{X}). &
\end{array}$$

■

Theorem 3.5.7 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-denotational observable, A be an atom, $\mathbf{G}_1$, $\mathbf{G}_2$ be goals and P be a program. Then*

1. $\mathsf{A} \tilde{\cdot} \mathcal{O}_\alpha[\![\mathsf{P}]\!] \leq \mathcal{B}_\alpha[\![\mathsf{A} \textit{ in } \mathsf{P}]\!]$,

2. $\mathcal{B}_\alpha[\![\mathbf{G}_1 \textit{ in } \mathsf{P}]\!] \tilde{\times} \mathcal{B}_\alpha[\![\mathbf{G}_2 \textit{ in } \mathsf{P}]\!] \leq \mathcal{B}_\alpha[\![(\mathbf{G}_1, \mathbf{G}_2) \textit{ in } \mathsf{P}]\!]$.

Proof. We prove the points separately.

Point 1 Analogously to Lemma 3.5.5,

$$\mathcal{O}_\alpha[\![\mathsf{P}]\!] \leq \left[\widetilde{\sum}\left\{ unf^{k}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))\right\}_{k\geq 0}\right] /_{\equiv_{\mathbb{A}}}. \tag{1}$$

Then the following facts hold.

$$\begin{aligned}
&\mathsf{A} \tilde{\cdot} \mathcal{O}_\alpha[\![\mathsf{P}]\!] \leq && [\text{by } (1)]\\
&\mathsf{A} \tilde{\cdot} \left[\widetilde{\sum}\{unf^{k}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))\}_{k\geq 0}\right] /_{\equiv_{\mathbb{A}}} = && [\text{by } (3.5.1)]\\
&\widetilde{\sum}\{\mathsf{A} \tilde{\cdot} unf^{k}_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))\}_{k\geq 0} \leq && [\text{by Point 1 of Proposition 3.5.6}]\\
&\widetilde{\sum}\{unf^{k}_{\mathsf{P},\alpha}(\mathsf{A} \tilde{\cdot} \alpha(Id_{\mathbb{I}}))\}_{k\geq 0} = && [\text{by } (3.5.2) \text{ and definition of } \tilde{\cdot}]\\
&\widetilde{\sum}\{unf^{k}_{\mathsf{P},\alpha}(\alpha(\phi_{\mathsf{A}}))\}_{k\geq 0} = && [\text{by } (3.2.17)]\\
&\mathcal{B}_\alpha[\![\mathsf{A}\ in\ \mathsf{P}]\!].
\end{aligned}$$

Point 2

$$\begin{aligned}
&\mathcal{B}_\alpha[\![\mathbf{G}_1\ in\ \mathsf{P}]\!] \tilde{\times} \mathcal{B}_\alpha[\![\mathbf{G}_2\ in\ \mathsf{P}]\!] =\\
&\qquad [\text{by } (3.2.17) \text{ and by } (3.5.1)]\\
&\widetilde{\sum}\{unf^{k}_{\mathsf{P},\alpha}(\alpha(\phi_{\mathbf{G}_1})) \tilde{\times} unf^{h}_{\mathsf{P},\alpha}(\alpha(\phi_{\mathbf{G}_2}))\}_{h,k\geq 0} \leq\\
&\qquad [\text{by using repeatedly Point 3 of Proposition 3.5.6}]\\
&\widetilde{\sum}\{unf^{h}_{\mathsf{P},\alpha}(unf^{k}_{\mathsf{P},\alpha}(\alpha(\phi_{\mathbf{G}_1})) \tilde{\times} \alpha(\phi_{\mathbf{G}_2}))\}_{h,k\geq 0} \leq\\
&\qquad [\text{by using repeatedly Point 2 of Proposition 3.5.6}]\\
&\widetilde{\sum}\{unf^{h}_{\mathsf{P},\alpha}(unf^{k}_{\mathsf{P},\alpha}(\alpha(\phi_{\mathbf{G}_1}) \tilde{\times} \alpha(\phi_{\mathbf{G}_2})))\}_{h,k\geq 0} =\\
&\qquad [\text{by definition of } unf_{\mathsf{P},\alpha}]\\
&\widetilde{\sum}\{unf^{k}_{\mathsf{P},\alpha}(\alpha(\phi_{\mathbf{G}_1}) \tilde{\times} \alpha(\phi_{\mathbf{G}_2}))\}_{k\geq 0} =\\
&\qquad [\text{by definition of } \tilde{\times} \text{ and } \times, \text{ by } (3.5.4) \text{ and } (3.2.17)]\\
&\mathcal{B}_\alpha[\![(\mathbf{G}_1, \mathbf{G}_2)\ in\ \mathsf{P}]\!].
\end{aligned}$$

■

Corollary 3.5.8 *Let α be a semi-denotational observable. Then*

$$\alpha(\mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!]) \leq \mathcal{Q}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!] \leq \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!].$$

Proof. The proof is by structural induction on $\mathbf{G}$.

$\underline{\mathbf{G} = \square}$ $\mathcal{Q}_\alpha[\![\square \ in \ \mathsf{P}]\!] = \mathcal{G}_\alpha[\![\square]\!]_{\mathcal{F}_\alpha[\![\mathsf{P}]\!]} = \alpha(\phi_\square) = \mathcal{B}_\alpha[\![\square \ in \ \mathsf{P}]\!]$, by definition of $\mathcal{Q}_\alpha[\![\cdot]\!]$, $\mathcal{G}_\alpha[\![\cdot]\!]$ and $\mathcal{B}_\alpha[\![\cdot]\!]$.

$\underline{\mathbf{G} = \mathbf{A}, \mathbf{G}'}$ The following facts hold.

$$
\begin{array}{ll}
\mathcal{Q}_\alpha[\![\mathbf{G} \ in \ \mathsf{P}]\!] = & [\text{by definition of } \mathcal{Q}_\alpha[\![\cdot]\!] \text{ and } \mathcal{F}_\alpha[\![\cdot]\!]] \\
\mathcal{G}_\alpha[\![(\mathsf{A}, \mathbf{G}')]\!]_{\mathcal{F}_\alpha[\![\mathsf{P}]\!]} = & [\text{by definition of } \mathcal{G}_\alpha[\![\cdot]\!] \text{ and } \mathcal{Q}_\alpha[\![\cdot]\!]] \\
\mathcal{A}_\alpha[\![\mathsf{A}]\!]_{\mathcal{F}_\alpha[\![\mathsf{P}]\!]} \widetilde{\times} \mathcal{Q}_\alpha[\![\mathbf{G}' \ in \ \mathsf{P}]\!] \leq & [\text{by inductive hypothesis}] \\
\mathcal{A}_\alpha[\![\mathsf{A}]\!]_{\mathcal{F}_\alpha[\![\mathsf{P}]\!]} \widetilde{\times} \mathcal{B}_\alpha[\![\mathbf{G}' \ in \ \mathsf{P}]\!] = & [\text{by definition of } \mathcal{A}_\alpha[\![\cdot]\!]] \\
(\mathsf{A} \tilde{\cdot} \mathcal{F}_\alpha[\![\mathsf{P}]\!]) \widetilde{\times} \mathcal{B}_\alpha[\![\mathbf{G}' \ in \ \mathsf{P}]\!] \leq & [\text{by Lemma 3.5.5}] \\
(\mathsf{A} \tilde{\cdot} \mathcal{O}_\alpha[\![\mathsf{P}]\!]) \widetilde{\times} \mathcal{B}_\alpha[\![\mathbf{G}' \ in \ \mathsf{P}]\!] \leq & [\text{by Point 1 of Theorem 3.5.7}] \\
\mathcal{B}_\alpha[\![\mathsf{A} \ in \ \mathsf{P}]\!] \widetilde{\times} \mathcal{B}_\alpha[\![\mathbf{G}' \ in \ \mathsf{P}]\!] \leq & [\text{by Point 2 of Theorem 3.5.7}] \\
\mathcal{B}_\alpha[\![\mathbf{G} \ in \ \mathsf{P}]\!]. &
\end{array}
$$

■

3.5.1 The Observable ψ for Groundness Analysis of Computed Answers

We show now how to obtain *Groundness analysis of computed answers* for pure logic programs [8, 79, 31] by applying our scheme. In order to define the abstract domain we have to do several small steps. We will use propositional formulas to represent the groundness dependencies of variables. In particular, we will use the domain $\mathcal{POS}$ [8] of positive propositional formulas classes modulo logical equivalence, built using $\leftrightarrow$, $\wedge$ and $\vee$, ordered by logical implication.

First of all we have to define the abstraction $\Gamma(\mathsf{t})$ of a concrete term t. If $var(\mathsf{t}) = \{\mathsf{x}_1, \ldots, \mathsf{x}_n\}$ then $\Gamma(\mathsf{t}) := \mathsf{x}_1 \wedge \cdots \wedge \mathsf{x}_n$, while if t is ground $\Gamma(\mathsf{t}) := \mathit{true}$. The formula intuitively suggests that in order for t to be ground, all its variables $\mathsf{x}_1, \ldots, \mathsf{x}_n$ must be ground.

We can extend Γ to substitutions to obtain abstract substitutions as

$$\Gamma(\vartheta) := \bigwedge_{\mathsf{x}/\mathsf{t} \in \vartheta} (\mathsf{x} \leftrightarrow \Gamma(\mathsf{t}))$$

where $\Gamma(\varepsilon) := \mathit{true}$. Abstract substitutions are propositional formulas which express the *groundness dependencies* between the variables of the domain and the ones of the range of the concrete substitution.

We must define the abstract notion of restriction of an abstract substitution w.r.t. a set of variables. Namely, by using *Schröder's elimination principle*,

$$\mathsf{F}|_{\mathbf{x}} := \begin{cases} \mathsf{F} & \text{if } var(\mathsf{F}) \subseteq \{\mathbf{x}\} \\ (\mathsf{F}[\mathsf{y} \mapsto \mathit{true}] \vee \mathsf{F}[\mathsf{y} \mapsto \mathit{false}])|_{\mathbf{x}} & \text{for some } \mathsf{y} \in var(\mathsf{F}) \setminus \{\mathbf{x}\} \end{cases}$$

where the formula $\mathsf{F}[\mathsf{y} \mapsto \mathsf{E}]$ is obtained by replacing each occurrence of the variable y in F by E.

Let $\mathbb{A}_{\mathrm{gr}} \subseteq [\mathit{Goals} \rightharpoonup \mathcal{POS}]$. We can obtain the desired abstraction ψ by further abstraction of computed answer substitutions. Namely $\tau_\Gamma : \mathbb{A}_{\mathrm{ca}} \to \mathbb{A}_{\mathrm{gr}}$ is defined as

$$\tau_\Gamma(\mathsf{X}) := \lambda \mathbf{G}. \bigvee_{\vartheta \in \mathsf{X}(\mathbf{G})} \Gamma(\vartheta)|_{var(\mathbf{G})},$$

where $\bigvee \emptyset := \mathit{false}$ (hence $\tau_\Gamma(\phi\left[{}^{\emptyset}/_{\mathsf{G}}\right]) = \phi\left[{}^{\mathit{false}}/_{\mathsf{G}}\right]$).

The *groundness dependencies of computed answer* observable $\psi : \mathbb{C} \to \mathbb{A}_{\mathrm{gr}}$ is $\psi := \tau_\Gamma \circ \xi$.

Example 3.5.9

Consider the $\mathbb{A}_{\mathrm{ca}}$-collections of Example 3.1.3 (on page 93). Since

$$\Gamma(\{\{\mathsf{x}/\mathsf{g}(\mathsf{a}), \mathsf{y}/\mathsf{b}\}\})\tilde{|}_{var(\mathsf{p}(\mathsf{x},\mathsf{y}))} = (\mathsf{x} \leftrightarrow \mathit{true} \wedge \mathsf{y} \leftrightarrow \mathit{true})\tilde{|}_{\mathsf{x},\mathsf{y}} = \mathsf{x} \wedge \mathsf{y},$$
$$\Gamma(\emptyset) = \mathit{false},$$

then

$$\psi(\mathsf{D}_1) = \tau_\Gamma(\xi(\mathsf{D}_1)) = \begin{cases} \mathsf{p}(\mathsf{x},\mathsf{y}) \mapsto \mathsf{x} \wedge \mathsf{y} \\ \mathsf{r}(\mathsf{x},\mathsf{y}) \mapsto \mathit{false} \end{cases}$$

Moreover, since $\Gamma(\{\mathsf{x}/\mathsf{g}(\mathsf{s}), \mathsf{z}/\mathsf{a}\})\tilde{|}_{\mathsf{x},\mathsf{z}} = (\mathsf{x} \leftrightarrow \mathsf{s} \wedge \mathsf{z})\tilde{|}_{\mathsf{x},\mathsf{z}} = \mathsf{z}$,

$$\psi(\mathsf{D}_2) = \begin{cases} \mathsf{q}(\mathsf{x},\mathsf{y}) & \mapsto \mathsf{x} \\ \mathsf{q}(\mathsf{f}(\mathsf{x}),\mathsf{z}) & \mapsto \mathsf{z} \end{cases}$$

Going further, since $\Gamma(\{\mathsf{x}/[\mathsf{a},\mathsf{l}], \mathsf{z}/[\mathsf{a},\mathsf{l},\mathsf{h}]\})\tilde{|}_{\mathsf{x},\mathsf{z}} = (\mathsf{x} \leftrightarrow \mathsf{l} \wedge \mathsf{z} \leftrightarrow \mathsf{l} \wedge \mathsf{h})\tilde{|}_{\mathsf{x},\mathsf{z}} = (\mathsf{z} \leftrightarrow \mathsf{x} \wedge \mathsf{h})\tilde{|}_{\mathsf{x},\mathsf{z}} = \mathsf{z} \to \mathsf{x}$, then

$$\psi(\mathcal{Q}[\![\mathsf{G}\ \mathit{in}\ \mathsf{P}]\!]) = \mathsf{ap}([\mathsf{a}],[\mathsf{l}],\mathsf{x}), \mathsf{ap}(\mathsf{x},[\mathsf{h}],\mathsf{z}) \mapsto \mathsf{z} \to \mathsf{x}.$$

Moreover, since $\Gamma(\{\{\mathsf{x}/\mathsf{g}(\mathsf{a}), \mathsf{y}/\mathsf{b}\}, \{\mathsf{x}/\mathsf{a}\}\})\tilde{|}_{\mathsf{x},\mathsf{y}} = ((\mathsf{x} \wedge \mathsf{y}) \vee \mathsf{x})\tilde{|}_{\mathsf{x},\mathsf{y}} = \mathsf{x}$,

$$\psi(\mathsf{D}) = \begin{cases} \mathsf{p}(\mathsf{x},\mathsf{y}) \mapsto \mathsf{x} \\ \mathsf{n}(\mathsf{x}) & \mapsto \mathit{false} \end{cases}$$

To conclude, the abstraction of collection D'''' is $\psi(\mathsf{D}'''') = \mathsf{anc}(\mathsf{x},\mathsf{y}) \mapsto \mathsf{x} \wedge \mathsf{y}$.

By applying the definition[10] the abstract operators are

$$\begin{aligned}
A \tilde{\cdot} X &= \phi\left[{}^{F}/_{A}\right] \mid F := \bigvee \big\{ F'' \mid \langle H, F' \rangle \text{ is a renamed apart (w.r.t. } A) \\
&\qquad \text{version of } \langle A', X(A') \rangle, \text{ for some } A' \leq A, \text{ there exists } \delta \text{ s.t.} \\
&\qquad A = H\delta,\ dom(\delta) \subseteq var(H) \text{ and } F'' = (F' \wedge \Gamma(\delta))|_{var(A)} \big\} \\
X_1 \tilde{\times} X_2 &= \lambda \mathbf{G}.\ \bigvee \big\{ X_1'(\mathbf{G}_1) \wedge X_2'(\mathbf{G}_2) \mid \mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2) \text{ and } X_1', X_2' \text{ are} \\
&\qquad \text{renamed versions of } X_1, X_2 \text{ s.t. } X_1'(\mathbf{G}_1) \text{ and } X_2'(\mathbf{G}_2) \text{ are defined} \big\} \\
\widetilde{\sum} \{X_i\}_{i \in I} &= \lambda \mathbf{G}.\ \bigvee \{X_i(\mathbf{G})\}_{i \in I}.
\end{aligned}$$

Example 3.5.10

Consider the following collections in $\mathbb{A}_{gr}$.

$$X_1 := \begin{cases} s(y) & \mapsto y \\ p(x, y) & \mapsto \mathit{true} \end{cases} \qquad X_2 := \begin{cases} q(x, y) & \mapsto x \leftrightarrow y \\ s(y) & \mapsto \mathit{false} \end{cases}$$

Since $s(a) = s(y)\{y/a\}$ and $(y \wedge \Gamma(\{y/a\}))\tilde{|}_{\emptyset} = (y \wedge y \leftrightarrow \mathit{true})\tilde{|}_{\emptyset} = \mathit{true}$,

$$s(a) \tilde{\cdot} X_1 = s(a) \mapsto \mathit{true}$$

Moreover, since $s(f(a, y) = s(x))\{x/f(a, y)\}$ and $(x \wedge \Gamma(\{x/f(a, y)\}))\widetilde{|}_y = (x \wedge x \leftrightarrow y)\widetilde{|}_y = (x \wedge y)\widetilde{|}_y = y$,

$$s(f(a, y)) \tilde{\cdot} X_1 = s(f(a, y)) \mapsto y$$

Furthermore, the following hold.

$$X_1 \tilde{\times} X_2 = \begin{cases} s(y), q(x, y) & \mapsto x \wedge y \\ s(y), s(y) & \mapsto \mathit{false} \\ p(x, y), q(x, y) & \mapsto x \leftrightarrow y \\ p(x, y), s(y) & \mapsto \mathit{false} \end{cases} \qquad X_1 \tilde{+} X_2 = \begin{cases} s(y) & \mapsto y \\ p(x, y) & \mapsto \mathit{true} \\ q(x, y) & \mapsto x \leftrightarrow y \end{cases}$$

Note that $supp(X_1 \tilde{+} X_2) = supp(X_1) \cup supp(X_2)$ and $supp(X_1 \tilde{\times} X_2) = \{(\mathbf{G}_1, \mathbf{G}_2) \mid \mathbf{G}_1 \in supp(X_1), \mathbf{G}_2 \in supp(X_2)\}$.

[10] In Section 3.9 we will show how the abstract operations of ψ can be defined in terms of τ_Γ and the abstract operators of ξ.

```
r(X,Y) :- p(X,Y), q(X,Y).
p(a,Y).
p(X,b).
q(X,X).
```

Figure 3.2: The program P of Example 3.5.10

In Section 3.9 we prove that ψ is semi-denotational and that the optimal operator $\widetilde{\mathcal{C}}[\![\cdot]\!]$ is

$$\widetilde{\mathcal{C}}[\![p(\mathbf{t}) \leftarrow p_1(\mathbf{t}_1), \ldots, p_n(\mathbf{t}_n)]\!]_X = \lambda p(\mathbf{x}).\, (\mathbf{x} \leftrightarrow \Gamma(\mathbf{t}) \wedge \bigwedge_{i=1}^{n} (\mathbf{x}_i \leftrightarrow \Gamma(\mathbf{t}_i) \wedge X(p_i(\mathbf{x}_i))))|_{\mathbf{x}}$$

where, $\forall i \in [1, n]$, $\mathbf{x}$, $\mathbf{x}_i$ are new distinct variables, $X(p_i(\mathbf{x}_i))$ is defined and the formula $\mathbf{x} \leftrightarrow \Gamma(\mathbf{t})$ denotes $\bigwedge_{i=1}^{n}(x_i \leftrightarrow \Gamma(t_i))$ (given that $\mathbf{x} = x_1, \ldots, x_n$ and $\mathbf{t} = t_1, \ldots, t_n$), which is equivalent to $\Gamma(\{\mathbf{x}/\mathbf{t}\})$.

Example 3.5.11

Let P be the program of Figure 3.2, and Q the program of Figure 3.3. Consider the collections

$$I := \begin{cases} p(x,y) \mapsto \mathit{true} \\ q(x,y) \mapsto x \leftrightarrow y \\ r(x,y) \mapsto x \leftrightarrow y \end{cases} \qquad K := \begin{cases} p(x) \mapsto x \\ q(x) \mapsto x \\ r(x) \mapsto \mathit{false} \\ s(x,y) \mapsto y \end{cases}$$

For the first clause of P we obtain

$$\begin{aligned} \mathcal{P}_\psi[\![r(x,y) \leftarrow p(x,y), q(x,y)]\!]_I &= r(x,y) \mapsto I(p(x,y)) \vee I(q(x,y)) \\ &= r(x,y) \mapsto x \leftrightarrow y \end{aligned}$$

and

$$\mathcal{P}_\psi[\![P]\!]_I = \begin{cases} p(x,y) \mapsto x \vee y \\ q(x,y) \mapsto x \leftrightarrow y \\ r(x,y) \mapsto x \leftrightarrow y \end{cases} \qquad \mathcal{P}_\psi[\![Q]\!]_K = \begin{cases} p(x) \mapsto x \\ q(x) \mapsto x \\ r(x) \mapsto \mathit{false} \\ s(x,y) \mapsto y \end{cases}$$

```
p(f(X)) :- q(X).
q(a).
r(X) :- p(g(X)).
s(X,Y) :- r(X).
s(X,a).
```

Figure 3.3: The program Q of Example 3.5.10

We are guaranteed that we can finitely compute the fixpoint of P, since $\mathcal{POS}$ is Noetherian. Indeed,

$$\mathcal{P}_\psi[\![\mathsf{P}]\!]\uparrow_1 = \begin{cases} p(x,y) \mapsto x \vee y \\ q(x,y) \mapsto x \leftrightarrow y \\ r(x,y) \mapsto \mathit{false} \end{cases} \qquad \mathcal{P}_\psi[\![\mathsf{P}]\!]\uparrow_2 = \begin{cases} p(x,y) \mapsto x \vee y \\ q(x,y) \mapsto x \leftrightarrow y \\ r(x,y) \mapsto x \wedge y \end{cases}$$

and $\mathcal{P}_\psi[\![\mathsf{P}]\!]\uparrow_3 = \mathcal{P}_\psi[\![\mathsf{P}]\!]\uparrow_2 = \mathcal{F}_\psi[\![\mathsf{P}]\!]$. Moreover,

$$\mathcal{P}_\psi[\![\mathsf{Q}]\!]\uparrow_1 = \begin{cases} p(x) \mapsto \mathit{false} \\ q(x) \mapsto x \\ r(x) \mapsto \mathit{false} \\ s(x,y) \mapsto y \end{cases} \qquad \mathcal{P}_\psi[\![\mathsf{Q}]\!]\uparrow_2 = \begin{cases} p(x) \mapsto x \\ q(x) \mapsto x \\ r(x) \mapsto \mathit{false} \\ s(x,y) \mapsto y \end{cases}$$

$$\mathcal{P}_\psi[\![\mathsf{Q}]\!]\uparrow_3 = \begin{cases} p(x) \mapsto x \\ q(x) \mapsto x \\ r(x) \mapsto x \\ s(x,y) \mapsto y \end{cases} \qquad \mathcal{P}_\psi[\![\mathsf{Q}]\!]\uparrow_4 = \begin{cases} p(x) \mapsto x \\ q(x) \mapsto x \\ r(x) \mapsto x \\ s(x,y) \mapsto x \vee y \end{cases}$$

and $\mathcal{P}_\psi[\![\mathsf{Q}]\!]\uparrow_5 = \mathcal{P}_\psi[\![\mathsf{Q}]\!]\uparrow_4 = \mathcal{F}_\psi[\![\mathsf{Q}]\!]$, while

$$\psi(\mathcal{F}[\![\mathsf{Q}]\!]) = \begin{cases} p(x) \mapsto x \\ q(x) \mapsto x \\ r(x) \mapsto \mathit{false} \\ s(x,y) \mapsto y \end{cases}$$

This shows the loss of information due to ψ, since there is no concrete computation for $r(x)$, while we derive an abstract computed answer for it.

A Noetherian domain which looses less information than $\mathcal{POS}$ is shown in the next section.

3.5.2 The *depth*(k) Observable

Now we show how to approximate an infinite set of computed answers by means of a *depth*(k) cut [95], i.e., by cutting terms which have a depth greater than k. Terms are cut by replacing each sub-term rooted at depth k with a new variable taken from a set $\hat{V}$ (disjoint from V). *depth*(k) terms represent each term obtained by instantiating the variables of $\hat{V}$ with terms built over V.

First of all we have to define the abstraction $t\lightning_k$ as the *depth*(k) reduction of the concrete term t.

We can extend $\lightning_k$ to substitutions to obtain abstract substitutions as $\vartheta\lightning_k := \{x/t\lightning_k \mid x/t \in \vartheta\}$. We assume that for any binding in ϑ any cut is performed by using distinct variables of $\hat{V}$. We denote by $Subst\lightning_k$ the set of substitutions $V \to T\lightning_k$, where $T\lightning_k$ is the set of *depth*(k) terms.

Let $\mathbb{A}_{\lightning_k} \subseteq [Goals \rightharpoonup \wp(Subst\lightning_k)]$. As in Section 3.5.1 we can obtain the *depth*(k) observable κ by further abstraction of computed answer substitutions. Namely $\mu_k : \mathbb{A}_{ca} \to \mathbb{A}_{\lightning_k}$ is defined as

$$\mu_k(X) := \lambda \mathbf{G}.\, \big\{\vartheta\lightning_k \mid \vartheta \in X(\mathbf{G})\big\}.$$

The *depth*(k) *computed answer* observable $\kappa : \mathbb{C} \to \mathbb{A}_{\lightning_k}$ is $\kappa := \mu_k \circ \xi$. For simplicity, in the following we assume that for any syntactic expression we rename variables in V and $\hat{V}$ respectively with variables still in V and $\hat{V}$. Then, by applying the definition[11], the abstract operators turn out to be

$$\begin{aligned} A \tilde{\cdot} X = \phi\big[{}^{\Theta}/_{A}\big] \text{ where } \Theta := \big\{\vartheta\lightning_k \mid \langle H, \Theta'\rangle &\text{ is a renamed apart} \\ &\text{(from } A\text{) version of } \langle A', X(A')\rangle, \text{ for some } A' \leq A,\ \vartheta' \in \Theta' \text{ and} \\ &\vartheta = mgu(A, H\vartheta')|_A\big\}, \\ X_1 \tilde{\times} X_2 = \lambda \mathbf{G}.\, \big\{\vartheta\lightning_k \mid \mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2), &\text{ for } i = 1, 2,\ \mathbf{G}'_i \equiv \mathbf{G}_i,\ \vartheta_i \text{ is a} \\ &\text{renamed version of an element in } X_i(\mathbf{G}'_i), \text{ via a renaming } \rho_i \text{ s.t.} \\ &\mathbf{G}'_i\rho_i = \mathbf{G}_i,\ var(\vartheta_1) \cap var(\vartheta_2) \subseteq var(\mathbf{G}_1) \cap var(\mathbf{G}_2) \text{ and} \\ &\vartheta = (\vartheta_1 \circ mgu(\mathbf{G}_2\vartheta_1, \mathbf{G}_2\vartheta_2))|_G\big\}. \end{aligned}$$

In Section 3.9 we prove that κ is semi-denotational and that the abstract semantic function is

$$\begin{aligned} \tilde{\mathcal{C}}[\![p(\mathbf{t}) \leftarrow \mathbf{B}]\!]_X = \lambda p(\mathbf{x}).\, \big\{\vartheta\lightning_k \mid \mathbf{x} &\text{ are new variables, } \vartheta_i \in X(p_i(\mathbf{x}_i)) \text{ and} \\ &\vartheta = (\{\mathbf{x}/\mathbf{t}\} \circ mgu(\mathbf{B}, (p_1(\mathbf{x}_1)\vartheta_1, \ldots, p_n(\mathbf{x}_n)\vartheta_n)))|_{\mathbf{x}}\big\}. \end{aligned}$$

[11] In Section 3.9 we will show how the abstract operations of κ can be defined in terms of μ_k and the abstract operators of ξ.

3.6 Semi-perfect Observables

The relation between operational and denotational definitions that we have noticed for precise (perfect and denotational) observables, holds for non-precise observables too. However, since for non-precise observables we only have a good denotational definition (semi-denotational observables), we should introduce a further class (semi-perfect observables). Semi-perfect observables have all the properties of perfect observables apart from precision. In particular, they have equivalent operational and denotational semantics, and the (top-down and bottom-up) goal-independent denotations are *AND*-compositional and *OR*-compositional. Let us just note that semi-perfect observables are essentially the observables which model top-down abstract interpretation frameworks (see, for example, [15]).

We relax the optimality conditions of perfect observables axioms to admit non-precise $\tilde{\cdot}$ $\tilde{\times}$ and $\tilde{\rhd}$ operators. However, we guarantee that weak[12] (compositionality) properties are still satisfied. Hence every perfect observable is a semi-perfect observable, but the converse does not hold.

Definition 3.6.1 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be an observable. Then α is a* semi-perfect *observable if, for any $\mathsf{A} \in$ Atoms, $\mathsf{D}', \mathsf{D}'' \in \mathbb{C}$, $\mathsf{D} \in \mathbb{PC}$ and $\mathbf{G} \in$ Goals the following properties hold.*

$$\tilde{\cdot},\ \tilde{\times} \text{ and } \tilde{\rhd} \text{ distribute over } \widetilde{\sum}, \tag{3.6.1}$$

$$\alpha(\mathsf{A} \cdot \gamma\alpha(Id_{\mathbb{I}})) = \alpha(\mathsf{A} \cdot Id_{\mathbb{I}}) = \alpha(\phi_{\mathsf{A}}), \tag{3.6.2}$$

$$\alpha(\gamma\alpha(\mathsf{D}') \rhd \gamma\alpha(Id_{\mathbb{C}})) = \alpha(\mathsf{D}' \rhd Id_{\mathbb{C}}) = \alpha(\mathsf{D}'), \tag{3.6.3}$$

$$\alpha(\gamma\alpha(\mathsf{D}') \times \gamma\alpha(\phi_{\mathbf{G}})) = \alpha(\mathsf{D}' \times \phi_{\mathbf{G}}), \tag{3.6.4}$$

$$\alpha(\gamma\alpha(\mathsf{D}') \rhd \gamma\alpha(su(\gamma\alpha(\mathsf{D})))) = \alpha(\mathsf{D}' \rhd su(\gamma\alpha(\mathsf{D}))), \tag{3.6.5}$$

$$\alpha(\gamma\alpha(\mathsf{D}') \rhd \gamma\alpha(\mathcal{G}[\![\mathbf{G}]\!]_{\gamma\alpha(\mathsf{D})})) = \alpha(\mathsf{D}' \rhd \mathcal{G}[\![\mathbf{G}]\!]_{\gamma\alpha(\mathsf{D})}), \tag{3.6.6}$$

$$\begin{aligned}&\alpha(\gamma\alpha(\mathsf{A} \cdot \gamma\alpha(\mathsf{D}')) \times \gamma\alpha(\mathcal{G}[\![\mathbf{G}]\!]_{\gamma\alpha(\mathsf{D})})) = \\ &\qquad \alpha((\mathsf{A} \cdot \gamma\alpha(\mathsf{D}')) \times \mathcal{G}[\![\mathbf{G}]\!]_{\gamma\alpha(\mathsf{D})}),\end{aligned} \tag{3.6.7}$$

$$\begin{aligned}&\alpha(\gamma\alpha(\mathsf{D}') \times \gamma\alpha(\gamma\alpha(\mathsf{D}'') \rhd su(\gamma\alpha(\mathsf{D})))) = \\ &\qquad \alpha(\gamma\alpha(\mathsf{D}') \times (\gamma\alpha(\mathsf{D}'') \rhd su(\gamma\alpha(\mathsf{D})))),\end{aligned} \tag{3.6.8}$$

$$\begin{aligned}&\alpha(\gamma\alpha(\mathsf{D}') \rhd \gamma\alpha(\gamma\alpha(\mathsf{D}'') \rhd su(\gamma\alpha(\mathsf{D})))) = \\ &\qquad \alpha(\gamma\alpha(\mathsf{D}') \rhd (\gamma\alpha(\mathsf{D}'') \rhd su(\gamma\alpha(\mathsf{D})))),\end{aligned} \tag{3.6.9}$$

$$\alpha(\mathsf{A} \cdot \gamma\alpha(\gamma\alpha(\mathsf{D}') \rhd su(\gamma\alpha(\mathsf{D})))) = \alpha(\mathsf{A} \cdot (\gamma\alpha(\mathsf{D}') \rhd su(\gamma\alpha(\mathsf{D})))). \tag{3.6.10}$$

[12] By weak properties we mean that we cannot ensure the precise properties of Sections 3.3 and 3.4, but their approximated formulation, where we replace the equality symbol by inequality. For example, $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!]$ becomes $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) \leq \mathcal{F}_\alpha[\![\mathsf{P}]\!]$.

First of all, we prove that every semi-perfect observable is also semi-denotational. By Definitions 3.6.1 and 3.5.1, we have only to show that equation (3.5.6) holds, namely $\alpha(\mathsf{D} \triangleright pu(\gamma\alpha(\sum\{\mathsf{D}_j\}_{j\in J}))) = \alpha(\mathsf{D} \triangleright pu(\sum\{\gamma\alpha(\mathsf{D}_j)\}_{j\in J}))$, for any $\mathsf{D} \in \mathbb{C}$ and every chain $\{\mathsf{D}_j\}_{j\in J} \subseteq \mathbb{PC}$.

Theorem 3.6.2 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-perfect observable, $\mathsf{X} \in \mathbb{PA}$, $\mathsf{D}' \in \mathbb{C}$, $\mathsf{D} \in \mathbb{PC}$, $\mathbf{G}$ be a goal, $\{\mathsf{X}_i\}_{i\in I} \subseteq \mathbb{PA}$ and $\{\mathsf{D}_j\}_{j\in J} \subseteq \mathbb{PC}$ be chains. Then*

1. $\mathcal{G}_\alpha[\![\mathbf{G}]\!]_{\mathsf{X}} = \alpha(\mathcal{G}[\![\mathbf{G}]\!]_{\gamma(\mathsf{X})})$,
2. $\mathcal{G}_\alpha[\![\mathbf{G}]\!]_{\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I}} = \widetilde{\sum}\{\mathcal{G}_\alpha[\![\mathbf{G}]\!]_{\mathsf{X}_i}\}_{i\in I}$,
3. $\alpha(\mathsf{D}' \triangleright pu(\gamma\alpha(\sum\{\mathsf{D}_j\}_{j\in J}))) = \alpha(\mathsf{D}' \triangleright pu(\sum\{\gamma\alpha(\mathsf{D}_j)\}_{j\in J}))$,
4. $\alpha(\mathsf{D}' \triangleright pu(\gamma\alpha(\mathsf{D}))) = \alpha(\gamma\alpha(\mathsf{D}') \triangleright pu(\gamma\alpha(\mathsf{D})))$.

Proof. We prove the points separately.

Point 1 The proof is by induction on $\mathbf{G}$. If $\mathbf{G} = \square$, by definition of $\mathcal{G}[\![\cdot]\!]$ and $\mathcal{G}_\alpha[\![\cdot]\!]$, $\mathcal{G}_\alpha[\![\square]\!]_{\mathsf{X}} = \alpha(\phi_\square) = \alpha(\mathcal{G}[\![\square]\!]_{\gamma(\mathsf{X})})$.

Otherwise let $\mathbf{G} = (\mathsf{A}, \mathbf{G}')$. The following facts hold.

$$
\begin{aligned}
&\mathcal{G}_\alpha[\![(\mathsf{A}, \mathbf{G}')]\!]_{\mathsf{X}} = \\
&\qquad [\text{by definition of } \mathcal{G}_\alpha[\![\cdot]\!]] \\
&\mathcal{A}_\alpha[\![\mathsf{A}]\!]_{\mathsf{X}} \mathbin{\widetilde{\times}} \mathcal{G}_\alpha[\![\mathbf{G}']\!]_{\mathsf{X}} = \\
&\qquad [\text{by definition of } \mathcal{A}_\alpha[\![\cdot]\!] \text{ and by inductive hypothesis}] \\
&(\mathsf{A} \mathbin{\widetilde{\cdot}} \mathsf{X}) \mathbin{\widetilde{\times}} \alpha(\mathcal{G}[\![\mathbf{G}']\!]_{\gamma(\mathsf{X})}) = \\
&\qquad [\text{by definition of } \widetilde{\cdot} \text{ and } \widetilde{\times}] \\
&\alpha(\gamma\alpha(\mathsf{A} \cdot \gamma(\mathsf{X})) \times \gamma\alpha(\mathcal{G}[\![\mathbf{G}']\!]_{\gamma(\mathsf{X})})) = \\
&\qquad [\text{by } (3.6.7)] \\
&\alpha((\mathsf{A} \cdot \gamma(\mathsf{X})) \times \mathcal{G}[\![\mathbf{G}']\!]_{\gamma(\mathsf{X})}) = \\
&\qquad [\text{by definition of } \mathcal{G}[\![\cdot]\!]] \\
&\alpha(\mathcal{G}[\![(\mathsf{A}, \mathbf{G}')]\!]_{\gamma(\mathsf{X})}).
\end{aligned}
$$

Point 2 The proof is by induction on $\mathbf{G}$. If $\mathbf{G} = \square$, by definition of $\mathcal{G}_\alpha[\![\cdot]\!]$, $\mathcal{G}_\alpha[\![\square]\!]_{\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I}} = \alpha(\phi_\square) = \widetilde{\sum}\{\mathcal{G}_\alpha[\![\square]\!]_{\mathsf{X}_i}\}_{i\in I}$.

Otherwise let $\mathbf{G} = (\mathsf{A}, \mathbf{G}')$. The following equivalences hold.

$$
\begin{aligned}
&\mathcal{G}_\alpha[\![(\mathsf{A}, \mathbf{G}')]\!]_{\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I}} = \\
&\qquad [\text{by definition of } \mathcal{G}_\alpha[\![\cdot]\!]]
\end{aligned}
$$

$$\mathcal{A}_\alpha[\![A]\!]_{\widetilde{\sum}\{X_i\}_{i\in I}} \mathbin{\widetilde{\times}} \mathcal{G}_\alpha[\![\mathbf{G}']\!]_{\widetilde{\sum}\{X_i\}_{i\in I}} =$$

[by definition of $\mathcal{A}_\alpha[\![\cdot]\!]$ and by inductive hypothesis]

$$(A \mathbin{\tilde{\cdot}} \widetilde{\sum}\{X_i\}_{i\in I}) \mathbin{\widetilde{\times}} \widetilde{\sum}\{\mathcal{G}_\alpha[\![\mathbf{G}']\!]_{X_i}\}_{i\in I} =$$

[by (3.6.1)]

$$\widetilde{\sum}\{(A \mathbin{\tilde{\cdot}} X_j) \mathbin{\widetilde{\times}} \mathcal{G}_\alpha[\![\mathbf{G}']\!]_{X_i}\}_{i,j\in I} =$$

[since $\{X_i\}_{i\in I}$ is a chain]

$$\widetilde{\sum}\{(A \mathbin{\tilde{\cdot}} X_i) \mathbin{\widetilde{\times}} \mathcal{G}_\alpha[\![\mathbf{G}']\!]_{X_i}\}_{i\in I} =$$

[by definition of $\mathcal{A}_\alpha[\![\cdot]\!]$ and $\mathcal{G}_\alpha[\![\cdot]\!]$]

$$\widetilde{\sum}\{\mathcal{G}_\alpha[\![(A, \mathbf{G}')]\!]_{X_i}\}_{i\in I}.$$

Point 3

$$\alpha(D' \triangleright pu(\gamma\alpha(\sum\{D_j\}_{j\in J}))) =$$

[by definition of pu and by Lemma 2.2.7]

$$\alpha(\sum\{D' \triangleright \mathcal{G}[\![\mathbf{G}]\!]_{\gamma\alpha(\sum\{D_j\}_{j\in J})}\}_{\mathbf{G}\in Goals}) =$$

[by (3.3.1) and (3.6.6)]

$$\alpha(\sum\{\gamma\alpha(\gamma\alpha(D') \triangleright \gamma\alpha(\mathcal{G}[\![\mathbf{G}]\!]_{\gamma\alpha(\sum\{\gamma\alpha(D_j)\}_{j\in J})}))\}_{\mathbf{G}\in Goals}) =$$

[by definition of the abstract operators]

$$\widetilde{\sum}\{\alpha(D') \mathbin{\widetilde{\triangleright}} \alpha(\mathcal{G}[\![\mathbf{G}]\!]_{\gamma(\widetilde{\sum}\{\alpha(D_j)\}_{j\in J})})\}_{\mathbf{G}\in Goals} =$$

[by Point 1]

$$\widetilde{\sum}\{\alpha(D') \mathbin{\widetilde{\triangleright}} \mathcal{G}_\alpha[\![\mathbf{G}]\!]_{\widetilde{\sum}\{\alpha(D_j)\}_{j\in J}}\}_{\mathbf{G}\in Goals} =$$

[by Point 2 and by (3.6.1)]

$$\widetilde{\sum}\{\widetilde{\sum}\{\alpha(D') \mathbin{\widetilde{\triangleright}} \mathcal{G}_\alpha[\![\mathbf{G}]\!]_{\alpha(D_j)}\}_{j\in J}\}_{\mathbf{G}\in Goals} =$$

[by definition of the abstract operators and by Point 1]

$$\alpha(\sum\{\gamma\alpha(\sum\{\gamma\alpha(\gamma\alpha(D') \triangleright \gamma\alpha(\mathcal{G}[\![\mathbf{G}]\!]_{\gamma\alpha(D_j)}))\}_{j\in J})\}_{\mathbf{G}\in Goals}) =$$

[by (3.6.6) and (3.3.1)]

$$\alpha(\sum\{\sum\{D' \triangleright \mathcal{G}[\![\mathbf{G}]\!]_{\gamma\alpha(D_j)}\}_{j\in J}\}_{\mathbf{G}\in Goals}) =$$

[by Lemma 2.2.7 and since $\{D_j\}_{j\in J}$ is a chain]

$$\alpha(D' \triangleright \sum\{\mathcal{G}[\![\mathbf{G}]\!]_{\sum\{\gamma\alpha(D_j)\}_{j\in J}}\}_{\mathbf{G}\in Goals}) =$$

[by definition of pu]

$\alpha(D' \triangleright pu(\sum\{\gamma\alpha(D_j)\}_{j\in J}))$.

Point 4 The proof is straightforward, by definition of pu, by Lemma 2.2.7 and by (3.6.6). ■

Corollary 3.6.3 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-perfect observable. Then α is a semi-denotational observable.*

Lemma 3.6.4 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-perfect observable and let $X \in \mathbb{PA}$ and $\{X_j\}_{j\in J} \subseteq \mathbb{PA}$ be a chain.*

1. $su_\alpha(X) = \alpha(su(\gamma(X)))$,
2. $\alpha(su_n(\gamma(\widetilde{\sum}\{X_j\}_{j\in J}))) = \alpha(\sum\{su_n(\gamma(X_j))\}_{j\in J})$.

Proof. We prove the points separately.

Point 1 The proof follows by Corollary 3.6.3 and by Point 2 of Lemma 3.5.3.

Point 2 We prove the two inclusions separately.

$\geq$ Straightforward, since $\cdot$, $\times$, $\sum$ and α are monotonic, $\gamma(X_j) \sqsubseteq \gamma(\widetilde{\sum}\{X_j\}_{j\in J})$, for any $j \in J$, and $\sum$ is the *lub* operation on $\mathbb{C}$.

$\leq$ The proof is by induction on n. For $n = 0$ the proof is immediate, by definition of su_0. For $n > 0$ the following equivalences hold.

$$\alpha(su_n(\gamma(\widetilde{\sum}\{X_j\}_{j\in J}))) =$$

[by definition of su_n]

$$\alpha(su_{n-1}(\gamma(\widetilde{\sum}\{X_j\}_{j\in J})) \triangleright su(\gamma(\widetilde{\sum}\{X_j\}_{j\in J}))) \leq$$

[by inductive hypothesis and since $\gamma\alpha$ is extensive]

$$\alpha(\gamma\alpha(\sum\{su_{n-1}(\gamma(X_j))\}_{j\in J}) \triangleright \gamma\alpha(\sum\{su(\gamma(X_j))\}_{j\in J})) \leq$$

[by definition of $\widetilde{\sum}$ and $\widetilde{\triangleright}$ and since $\gamma\alpha$ is extensive]

$$\widetilde{\sum}\{\alpha(su_{n-1}(\gamma(X_j)))\}_{j\in J} \widetilde{\triangleright} \widetilde{\sum}\{\alpha(su(\gamma(X_j)))\}_{j\in J} =$$

[by (3.6.1) and since $\{X_j\}_{j\in J} \subseteq \mathbb{PA}$ is a chain]

$$\widetilde{\sum}\{\alpha(su_{n-1}(\gamma(X_j))) \widetilde{\triangleright} \alpha(su(\gamma(X_j)))\}_{j\in J} =$$

[by definition of the abstract operators]

$$\alpha(\sum\{\gamma\alpha(\gamma\alpha(su_{n-1}(\gamma(X_j))) \triangleright \gamma\alpha(su(\gamma(X_j))))\}_{j\in J}) =$$

[by (3.6.5) and (3.3.1)]

$$\alpha(\sum\{su_{n-1}(\gamma(X_j)) \triangleright su(\gamma(X_j))\}_{j\in J}) =$$
$$[\text{by definition of } su_n]$$
$$\alpha(\sum\{su_n(\gamma(X_j))\}_{j\in J}).$$

■

Lemma 3.6.5 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-perfect observable, P be a program and* **G** *be a goal. Then*

1. $\mathcal{B}_\alpha[\![\mathbf{G}\ in\ P]\!] = \alpha(\phi_G \triangleright \sum\{su_n(\gamma\alpha(tree(P)))\}_{n\geq 0})$.
2. $\mathcal{O}_\alpha[\![P]\!] = (\alpha(Id_{\mathbb{I}} \triangleright \sum\{su_n(\gamma\alpha(tree(P)))\}_{n\geq 0}))\,/_{\equiv_{\mathbb{A}}}$.

Proof. We prove the points separately.

Point 1 First of all, we prove that $unf^n_{P,\alpha}(\alpha(\phi_G)) = \alpha(\phi_G \triangleright su_n(\gamma\alpha(tree(P))))$. The proof is by induction on $n \geq 0$. For $n = 0$, by definition, $unf^0_{P,\alpha}(\alpha(\phi_G)) = \alpha(\phi_G) = \alpha(\phi_G \triangleright su_0(\gamma\alpha(tree(P))))$. For $n > 0$, the following equivalences hold.

$$unf^n_{P,\alpha}(\alpha(\phi_G)) =$$
$$[\text{by definition of } unf^n_{P,\alpha}]$$
$$unf^{n-1}_{P,\alpha}(\alpha(\phi_G)) \,\widetilde{\triangleright}\, su_\alpha(\alpha(tree(P))) =$$
$$[\text{by inductive hypothesis, definition of } \widetilde{\triangleright} \text{ and Point 1 of Lemma 3.6.4}]$$
$$\alpha(\gamma\alpha(\phi_G \triangleright su_{n-1}(\gamma\alpha(tree(P)))) \triangleright \gamma\alpha(su(\gamma\alpha(tree(P))))) =$$
$$[\text{by (3.6.5)}]$$
$$\alpha((\phi_G \triangleright su_{n-1}(\gamma\alpha(tree(P)))) \triangleright su(\gamma\alpha(tree(P)))) =$$
$$[\text{by Point 2 of Lemma 2.2.6}]$$
$$\alpha(\phi_G \triangleright su_n(\gamma\alpha(tree(P)))).$$

Therefore, the following equivalences hold.

$$\mathcal{B}_\alpha[\![\mathbf{G}\ in\ P]\!] = \qquad [\text{by (3.2.17)}]$$
$$\widetilde{\sum}\{unf^k_{P,\alpha}(\alpha(\phi_G))\}_{k\geq 0} = \qquad [\text{by the previous result and (3.3.1)}]$$
$$\alpha(\sum\{\phi_G \triangleright su_n(\gamma\alpha(tree(P)))\}_{n\geq 0}) = \qquad [\text{by Lemma 2.2.7}]$$
$$\alpha(\phi_G \triangleright \sum\{su_n(\gamma\alpha(tree(P)))\}_{n\geq 0}).$$

Point 2 The following facts hold.

$$\mathcal{O}_\alpha[\![P]\!] =$$
$$[\text{by definition}]$$

$$\widetilde{\sum}\{\mathcal{B}_\alpha[\![\mathsf{p}(\mathbf{x})\ \mathit{in}\ \mathsf{P}]\!]/_{\equiv_\mathbb{A}}\}_{\mathsf{p}(\mathbf{x})\in \mathit{Goals}} =$$

[since $\equiv_\mathbb{A}$ is a congruence w.r.t. $\widetilde{\sum}$]

$$\left(\widetilde{\sum}\{\mathcal{B}_\alpha[\![\mathsf{p}(\mathbf{x})\ \mathit{in}\ \mathsf{P}]\!]\}_{\mathsf{p}(\mathbf{x})\in \mathit{Goals}}\right)/_{\equiv_\mathbb{A}} =$$

[by the previous result]

$$\left(\widetilde{\sum}\{\alpha(\phi_{\mathsf{p}(\mathbf{x})} \triangleright \sum\{\mathit{su}_n(\gamma\alpha(\mathit{tree}(\mathsf{P})))\}_{n\geq 0})\}_{\mathsf{p}(\mathbf{x})\in \mathit{Goals}}\right)/_{\equiv_\mathbb{A}} =$$

[by (3.3.1) and definition of $\widetilde{\sum}$]

$$\left(\alpha(\sum\{\phi_{\mathsf{p}(\mathbf{x})} \triangleright \sum\{\mathit{su}_n(\gamma\alpha(\mathit{tree}(\mathsf{P})))\}_{n\geq 0}\}_{\mathsf{p}(\mathbf{x})\in \mathit{Goals}})\right)/_{\equiv_\mathbb{A}} =$$

[by Lemma 2.2.7]

$$\left(\alpha(\sum\{\phi_{\mathsf{p}(\mathbf{x})}\}_{\mathsf{p}(\mathbf{x})\in \mathit{Goals}} \triangleright \sum\{\mathit{su}_n(\gamma\alpha(\mathit{tree}(\mathsf{P})))\}_{n\geq 0})\right)/_{\equiv_\mathbb{A}} =$$

[by definition of $\mathit{Id}_\mathbb{I}$]

$$\left(\alpha(\mathit{Id}_\mathbb{I} \triangleright \sum\{\mathit{su}_n(\gamma\alpha(\mathit{tree}(\mathsf{P})))\}_{n\geq 0})\right)/_{\equiv_\mathbb{A}}.$$

■

We can always reconstruct the abstract behavior of a generic (non-pure and non-atomic) goal from the abstract behavior of pure atoms. Therefore, all the properties of *SLD*-trees stated in Chapter 2 hold for the abstract top-down denotation for any semi-perfect observable as well.

Theorem 3.6.6 *Let $\alpha : \mathbb{C} \to \mathbb{A}$ be a semi-perfect observable, A be an atom, $\mathbf{G}$, $\mathbf{G}'$ be goals and P be a program. Then*

1. $\mathcal{B}_\alpha[\![\mathsf{A}\ \mathit{in}\ \mathsf{P}]\!] = \mathsf{A} \widetilde{\cdot} \mathcal{O}_\alpha[\![\mathsf{P}]\!]$,
2. $\mathcal{B}_\alpha[\![(\mathbf{G}, \mathbf{G}')\ \mathit{in}\ \mathsf{P}]\!] = \mathcal{B}_\alpha[\![\mathbf{G}\ \mathit{in}\ \mathsf{P}]\!] \widetilde{\times} \mathcal{B}_\alpha[\![\mathbf{G}'\ \mathit{in}\ \mathsf{P}]\!]$.

Proof. We prove the points separately.

Point 1 We prove the two inclusions separately.

$\leq$ By Lemma Lemma 3.6.5, by definition of $\widetilde{\cdot}$ and since $\equiv_\mathbb{A}$ is a congruence w.r.t. $\widetilde{\cdot}$, we have to prove that

$$\alpha(\phi_\mathsf{A} \triangleright \sum\{\mathit{su}_n(\gamma\alpha(\mathit{tree}(\mathsf{P})))\}_{n\geq 0}) \leq \alpha(\mathsf{A} \cdot \gamma\alpha(\mathit{Id}_\mathbb{I} \triangleright \sum\{\mathit{su}_n(\gamma\alpha(\mathit{tree}(\mathsf{P})))\}_{n\geq 0})).$$

By using the same arguments of Point 1 of Theorem 2.5.4, we can prove that

$$\mathsf{A} \cdot (Id_{\mathbb{I}} \triangleright \sum\{su_n(\gamma\alpha(tree(\mathsf{P})))\}_{n\geq 0}) = \phi_{\mathsf{A}} \triangleright \sum\{su_n(\gamma\alpha(tree(\mathsf{P})))\}_{n\geq 0}.$$

Now the thesis follows by extensivity of $\gamma\alpha$ and by monotonicity of α and of $\cdot$.

$\geq$ The proof follows by Corollary 3.6.3 and Point 1 of Theorem 3.5.7.

Point 2 We prove the two inclusions separately.

$\leq$ By Point 1 of Lemma 3.6.5,

$$\mathcal{B}_\alpha[\![(\mathbf{G}, \mathbf{G}')\ in\ \mathsf{P}]\!] = \alpha(\phi_{(\mathbf{G},\mathbf{G}')} \triangleright \sum\{su_n(\gamma\alpha(tree(\mathsf{P})))\}_{n\geq 0}).$$

Then, by using the same arguments of Point 2 of Theorem 2.5.4, we can prove that

$$\phi_{(\mathbf{G},\mathbf{G}')} \triangleright \sum\{su_n(\gamma\alpha(tree(\mathsf{P})))\}_{n\geq 0} =$$
$$(\phi_{\mathbf{G}} \triangleright \sum\{su_n(\gamma\alpha(tree(\mathsf{P})))\}_{n\geq 0}) \times (\phi_{\mathbf{G}'} \triangleright \sum\{su_n(\gamma\alpha(tree(\mathsf{P})))\}_{n\geq 0}).$$

Now the thesis follows by observing that $\gamma\alpha$ is extensive, by Point 1 of Lemma 3.6.5 and by definition of $\widetilde{\times}$.

$\geq$ The proof follows by Corollary 3.6.3 and by Point 2 of Theorem 3.5.7. ■

From Theorem 3.6.6 we can immediately derive that, for any atom A, goal $\mathbf{G}$ and program P,

$$\mathcal{B}_\alpha[\![\square\ in\ \mathsf{P}]\!] = \phi_\square, \tag{3.6.11}$$
$$\mathcal{B}_\alpha[\![(\mathsf{A}, \mathbf{G})\ in\ \mathsf{P}]\!] = (\mathsf{A} \tilde{\cdot} \mathcal{O}_\alpha[\![\mathsf{P}]\!]) \widetilde{\times} \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!]. \tag{3.6.12}$$

Now we prove the abstract *OR*-compositionality for semi-perfect observables. First of all, by definition of $\widetilde{\uplus}$, we have to prove that the function $\widetilde{\mathcal{H}}_{\mathsf{X}}$ is continuous. Then, by construction, the least fixpoint of the function $\widetilde{\mathcal{H}}_{\mathsf{X}}$ is the least solution of the equation $[\mathsf{X}]^*_\alpha = \alpha(Id_{\mathbb{I}}) \widetilde{+} ([\mathsf{X}]^*_\alpha \widetilde{\triangleright} su_\alpha(\mathsf{X}))$.

Lemma 3.6.7 *Let $\mathsf{X} \in \mathbb{PA}$ and let α be a semi-perfect observable.*

1. $\widetilde{\mathcal{H}}_{\mathsf{X}} \circ \alpha = \alpha \circ \mathcal{H}_{\gamma(\mathsf{X})}$,
2. $\widetilde{\mathcal{H}}_{\mathsf{X}}$ *is continuous,*
3. $[\mathsf{X}]^*_\alpha = lfp_{\mathbb{A}}\, \widetilde{\mathcal{H}}_{\mathsf{X}} = \alpha(lfp_{\mathbb{C}}\, \mathcal{H}_{\gamma(\mathsf{X})}) = \alpha([\gamma(\mathsf{X})]^*)$.

Proof. We prove the points separately.

Point 1 Let $\mathsf{D} \in \mathbb{C}$. The following equivalences hold.

$$
\begin{array}{ll}
\widetilde{\mathcal{H}}_{\mathsf{X}}(\alpha(\mathsf{D})) = & [\text{by definition of } \widetilde{\mathcal{H}}_{\mathsf{X}}] \\
\alpha(Id_{\mathbb{I}}) \mathbin{\widetilde{+}} (\alpha(\mathsf{D}) \mathbin{\widetilde{\triangleright}} su_{\alpha}(\mathsf{X})) = & [\text{by Point 1 of Lemma 3.6.4}] \\
\alpha(Id_{\mathbb{I}}) \mathbin{\widetilde{+}} (\alpha(\mathsf{D}) \mathbin{\widetilde{\triangleright}} \alpha(su(\gamma(\mathsf{X})))) = & [\text{by definition of } \widetilde{\triangleright} \text{ and } \widetilde{+}] \\
\alpha(\gamma\alpha(Id_{\mathbb{I}}) + \gamma\alpha(\gamma\alpha(\mathsf{D}) \triangleright \gamma\alpha(su(\gamma(\mathsf{X}))))) = & [\text{by (3.6.5) and (3.3.1)}] \\
\alpha(Id_{\mathbb{I}} + (\mathsf{D} \triangleright su(\gamma(\mathsf{X})))) = & [\text{by definition of } \mathcal{H}_{\gamma(\mathsf{X})}] \\
\alpha(\mathcal{H}_{\gamma(\mathsf{X})}(\mathsf{D})). &
\end{array}
$$

Point 2 Let $\{\mathsf{X}_i\}_{i\in I} \subseteq \mathbb{PA}$ be a chain. Since $\widetilde{\sum}$ is the *lub* operation on $\mathbb{PA}$, we have to prove that $\widetilde{\sum}\{\widetilde{\mathcal{H}}_{\mathsf{X}}(\mathsf{X}_i)\}_{i\in I} = \widetilde{\mathcal{H}}_{\mathsf{X}}(\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I})$. The following equalities hold.

$$
\begin{array}{ll}
\widetilde{\mathcal{H}}_{\mathsf{X}}(\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I}) = & [\text{by definition of } \widetilde{\sum} \text{ and Point 1}] \\
\alpha(Id_{\mathbb{I}} + (\sum\{\gamma(\mathsf{X}_i)\}_{i\in I} \triangleright su(\gamma(\mathsf{X})))) = & [\text{by Lemma 2.2.7 and by (3.3.1)}] \\
\alpha(\sum\{\gamma\alpha(Id_{\mathbb{I}} + (\gamma(\mathsf{X}_i) \triangleright su(\gamma(\mathsf{X}))))\}_{i\in I}) = & [\text{by Point 1 and since } \alpha\gamma = Id] \\
\alpha(\sum\{\gamma(\widetilde{\mathcal{H}}_{\mathsf{X}}(\mathsf{X}_i))\}_{i\in I}) = & [\text{by definition of } \widetilde{\sum}] \\
\widetilde{\sum}\{\widetilde{\mathcal{H}}_{\mathsf{X}}(\mathsf{X}_i)\}_{i\in I}. &
\end{array}
$$

Point 3 First of all we prove that, for any $n \geq 0$, $\widetilde{\mathcal{H}}_{\mathsf{X}}{\uparrow}n = \alpha(\mathcal{H}_{\gamma(\mathsf{X})}{\uparrow}n)$. The proof is by induction on n. For $n = 0$ the proof is straightforward by observing that $\widetilde{\mathcal{H}}_{\mathsf{X}}{\uparrow}0 = \bot_{\mathbb{A}} = \alpha(\bot_{\mathbb{C}}) = \alpha(\mathcal{H}_{\gamma(\mathsf{X})}{\uparrow}0)$. For $n > 0$ the following equivalences hold.

$$
\begin{array}{ll}
\widetilde{\mathcal{H}}_{\mathsf{X}}{\uparrow}n = & [\text{by definition of } \cdot{\uparrow}n] \\
\widetilde{\mathcal{H}}_{\mathsf{X}}(\widetilde{\mathcal{H}}_{\mathsf{X}}{\uparrow}n-1) = & [\text{by inductive hypothesis}] \\
\widetilde{\mathcal{H}}_{\mathsf{X}}(\alpha(\mathcal{H}_{\gamma(\mathsf{X})}{\uparrow}n-1)) = & [\text{by Point 1}] \\
\alpha(\mathcal{H}_{\gamma(\mathsf{X})}(\mathcal{H}_{\gamma(\mathsf{X})}{\uparrow}n-1)) = & [\text{by definition of } \cdot{\uparrow}n] \\
\alpha(\mathcal{H}_{\gamma(\mathsf{X})}{\uparrow}n). &
\end{array}
$$

Finally

$$
\begin{array}{ll}
[\mathsf{X}]^*_{\alpha} = & [\text{by construction}] \\
\mathit{lfp}_{\mathbb{A}}\, \widetilde{\mathcal{H}}_{\mathsf{X}} = & [\text{by Point 2}] \\
\widetilde{\sum}\{\widetilde{\mathcal{H}}_{\mathsf{X}}{\uparrow}n\}_{n\geq 0} = & [\text{by the previous result}] \\
\widetilde{\sum}\{\alpha(\mathcal{H}_{\gamma(\mathsf{X})}{\uparrow}n)\}_{n\geq 0} = & [\text{by (3.3.1)}]
\end{array}
$$

$$\begin{array}{ll}
\alpha(\sum\{\mathcal{H}_{\gamma(X)}\uparrow n\}_{n\geq 0}) = & [\text{by continuity of } \mathcal{H}_{\gamma(X)}] \\
\alpha(lfp_{\mathbb{C}}\,\mathcal{H}_{\gamma(X)}) = & [\text{by construction}] \\
\alpha([\gamma(X)]^*). &
\end{array}$$

■

Corollary 3.6.8 *Let* $\alpha : \mathbb{C} \to \mathbb{A}$ *be a semi-perfect observable and* P_1, P_2 *be programs. Then* $\mathcal{O}_\alpha[\![P_1 \cup P_2]\!] = \mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{\uplus}\, \mathcal{O}_\alpha[\![P_2]\!]$.

Proof. First of all we prove that

$$\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{\uplus}\, \mathcal{O}_\alpha[\![P_2]\!] = \alpha(Id_{\mathbb{I}} \triangleright \sum \left\{ su_n(\gamma(\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!]))\right\}_{n\geq 0}). \tag{1}$$

The following equivalences hold.

$$\begin{array}{ll}
\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{\uplus}\, \mathcal{O}_\alpha[\![P_2]\!] = & [\text{by definition of } \widetilde{\uplus}] \\
[\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!]]^*_\alpha = & [\text{by Point 3 of Lemma 3.6.7}] \\
\alpha([\gamma(\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!])]^*) = & [\text{by the proof of Theorem 2.5.9}] \\
\alpha(Id_{\mathbb{I}} \triangleright \sum\{su_n(\gamma(\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!]))\}_{n\geq 0}). &
\end{array}$$

Now, we can prove the two inclusions separately.

$\leq$ By (1) and by Lemma 3.6.5, we have to prove that

$$\begin{array}{l}
\alpha(Id_{\mathbb{I}} \triangleright \sum\{su_n(\gamma\alpha(tree(P_1 \cup P_2)))\}_{n\geq 0}) \leq \\
\alpha(Id_{\mathbb{I}} \triangleright \sum\{su_n(\gamma(\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!]))\}_{n\geq 0}).
\end{array}$$

We prove that, for any $n \geq 0$, $su_n(\gamma\alpha(tree(P_1 \cup P_2))) \sqsubseteq su_n(\gamma(\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!]))$. Then the thesis follows by monotonicity of $\triangleright$ and α. Observe that, since (for any program P) $tree(P)$ is a pure collection, $\gamma\alpha(tree(P)) \sqsubseteq Id_{\mathbb{I}} \triangleright su(\gamma\alpha(tree(P))) \sqsubseteq \gamma(\mathcal{O}_\alpha[\![P]\!])$. Then, by definition of *tree* and $\widetilde{+}$ and by (3.3.1), $\gamma\alpha(tree(P_1 \cup P_2)) = \gamma(\alpha(tree(P_1)) \,\widetilde{+}\, \alpha(tree(P_2))) \sqsubseteq \gamma(\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!])$ and therefore, since $\cdot$ and $\times$ are monotonic, $su(\gamma\alpha(tree(P_1 \cup P_2))) \sqsubseteq su(\gamma(\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!]))$. Then, since $\triangleright$ is also monotonic, for any $n \geq 0$, $su_n(\gamma\alpha(tree(P_1 \cup P_2))) \sqsubseteq su_n(\gamma(\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!]))$.

$\geq$ First of all, observe that for any $i \in [1, 2]$, $\mathcal{O}_\alpha[\![P_i]\!] \leq \mathcal{O}_\alpha[\![P_1 \cup P_2]\!]$ and therefore, since $\widetilde{+}$ is the *lub* operation on $\mathbb{A}$, $\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!] \leq \mathcal{O}_\alpha[\![P_1 \cup P_2]\!]$. Then, since γ, $\cdot$ and $\times$ are monotonic, for any $n \geq 0$

$$su_n(\gamma(\mathcal{O}_\alpha[\![P_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![P_2]\!])) \sqsubseteq su_n(\gamma(\mathcal{O}_\alpha[\![P_1 \cup P_2]\!])). \tag{2}$$

Now the following statements hold.

$$
\begin{aligned}
& su_n(\gamma(\mathcal{O}_\alpha[\![\mathsf{P}_1 \cup \mathsf{P}_2]\!])) = \\
& \quad [\text{by Lemma 3.6.5}] \\
& su_n(\gamma\alpha(Id_{\mathbb{I}} \triangleright \sum\{su_n(\gamma\alpha(tree(\mathsf{P}_1 \cup \mathsf{P}_2)))\}_{n\geq 0})) \sqsubseteq \\
& \quad [\text{by definition of the abstract operators and by Lemma 2.2.7}] \\
& su_n(\gamma(\widetilde{\sum}\{unf^k_{\mathsf{P}_1\cup\mathsf{P}_2,\alpha}(\alpha(Id_{\mathbb{I}}))\}_{k\geq 0})) \sqsubseteq \\
& \quad [\text{by Point 2 of Lemma 3.6.4 and since } \gamma\alpha \text{ is extensive}] \\
& \gamma\alpha(\sum\{su_n(\gamma(unf^k_{\mathsf{P}_1\cup\mathsf{P}_2,\alpha}(\alpha(Id_{\mathbb{I}}))))\}_{k\geq 0}) \sqsubseteq \\
& \quad [\text{by Point 3 of Lemma 3.5.4 and Corollary 3.6.3}] \\
& \gamma\alpha(\sum\{\gamma(unf^{kn}_{\mathsf{P}_1\cup\mathsf{P}_2,\alpha}(\alpha(Id_{\mathbb{C}})))\}_{k\geq 0}) = \\
& \quad [\text{by set-theoretic properties}] \\
& \gamma\alpha(\sum\{\gamma(unf^k_{\mathsf{P}_1\cup\mathsf{P}_2,\alpha}(\alpha(Id_{\mathbb{C}})))\}_{k\geq 0}).
\end{aligned}
$$

Moreover, by (3.2.18) and (3.6.1), $\mathcal{O}_\alpha[\![\mathsf{P}]\!] = \left[\widetilde{\sum}\{unf^k_{\mathsf{P},\alpha}(\alpha(Id_{\mathbb{I}}))\}_{k\geq 0}\right] /_{\equiv_{\mathbb{A}}}$. Thus,

$$
\begin{aligned}
& \mathcal{O}_\alpha[\![\mathsf{P}_1]\!] \,\widetilde{\uplus}\, \mathcal{O}_\alpha[\![\mathsf{P}_2]\!] = \\
& \quad [\text{by (1)}] \\
& \alpha(Id_{\mathbb{I}} \triangleright \sum\{su_n(\gamma(\mathcal{O}_\alpha[\![\mathsf{P}_1]\!] \,\widetilde{+}\, \mathcal{O}_\alpha[\![\mathsf{P}_2]\!]))\}_{n\geq 0}) = \\
& \quad [\text{by (2), the previous result and since } \alpha \text{ is monotonic}] \\
& \alpha(Id_{\mathbb{I}} \triangleright \gamma\alpha(\sum\{\gamma(unf^k_{\mathsf{P}_1\cup\mathsf{P}_2,\alpha}(\alpha(Id_{\mathbb{C}})))\}_{k\geq 0})) \leq \\
& \quad [\text{by definition of the abstract operators and since } \gamma\alpha \text{ is extensive}] \\
& \alpha(Id_{\mathbb{I}}) \,\widetilde{\triangleright}\, \widetilde{\sum}\{unf^k_{\mathsf{P}_1\cup\mathsf{P}_2,\alpha}(\alpha(Id_{\mathbb{C}}))\}_{k\geq 0} \leq \\
& \quad [\text{by (3.6.1)}] \\
& \widetilde{\sum}\{\alpha(Id_{\mathbb{I}}) \,\widetilde{\triangleright}\, unf^k_{\mathsf{P}_1\cup\mathsf{P}_2,\alpha}(\alpha(Id_{\mathbb{C}}))\}_{k\geq 0} \leq \\
& \quad [\text{by Point 2 of Lemma 3.5.4 and Corollary 3.6.3}] \\
& \widetilde{\sum}\{unf^k_{\mathsf{P}_1\cup\mathsf{P}_2,\alpha}(\alpha(Id_{\mathbb{I}}))\}_{k\geq 0} = \\
& \quad [\text{by the previous observation}] \\
& \mathcal{O}_\alpha[\![\mathsf{P}_1 \cup \mathsf{P}_2]\!].
\end{aligned}
$$

■

In Corollary 3.6.10 we will prove that the abstract top-down and the abstract

bottom-up denotations are indeed equivalent, which implies (by Theorem 3.6.6) the equivalence between the denotational and the operational semantics.

Theorem 3.6.9 *Let* $\alpha : \mathbb{C} \to \mathbb{A}$ *be a semi-perfect observable,* c *be a clause,* $\mathsf{X} \in \mathsf{PA}$ *and* P *be a program. Then*

1. $\mathcal{C}_\alpha[\![\mathsf{c}]\!]_\mathsf{X} = \alpha(\mathit{tree}(\mathsf{c}) \triangleright \mathit{pu}(\gamma(\mathsf{X})))$,
2. $\mathcal{P}_\alpha[\![\mathsf{P}]\!]_\mathsf{X} = \alpha(\mathcal{P}[\![\mathsf{P}]\!]_{\gamma(\mathsf{X})})$,
3. $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ *is continuous on* $\mathbb{A}$ *and* $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathcal{P}_\alpha[\![\mathsf{P}]\!]\uparrow\omega$.

Proof. We prove the points separately.

Point 1 Let $\mathsf{c} = \mathsf{H} \leftarrow \mathbf{B}$. The following equalities hold.

$$
\begin{array}{ll}
\mathcal{C}_\alpha[\![\mathsf{c}]\!]_\mathsf{X} = & [\text{by definition of } \mathcal{C}_\alpha[\![\cdot]\!]\,] \\
\alpha(\mathit{tree}(\mathsf{c})) \,\widetilde{\triangleright}\, \mathcal{G}_\alpha[\![\mathbf{B}]\!]_\mathsf{X} = & [\text{by Point 1 of Theorem 3.6.2}] \\
\alpha(\mathit{tree}(\mathsf{c})) \,\widetilde{\triangleright}\, \alpha(\mathcal{G}[\![\mathbf{B}]\!]_{\gamma(\mathsf{X})}) = & [\text{by definition of } \widetilde{\triangleright}] \\
\alpha(\gamma\alpha(\mathit{tree}(\mathsf{c})) \triangleright \gamma\alpha(\mathcal{G}[\![\mathbf{B}]\!]_{\gamma(\mathsf{X})})) = & [\text{by (3.6.6)}] \\
\alpha(\mathit{tree}(\mathsf{c}) \triangleright \mathcal{G}[\![\mathbf{B}]\!]_{\gamma(\mathsf{X})}) = & [\text{by definition of } \mathit{pu}\,] \\
\alpha(\mathit{tree}(\mathsf{c}) \triangleright \mathit{pu}(\gamma(\mathsf{X}))). &
\end{array}
$$

Point 2 The proof is straightforward by Point 1, by (3.3.1) and by definition of $\mathcal{P}_\alpha[\![\cdot]\!]$ and is omitted.

Point 3 Let $\{\mathsf{X}_i\}_{i\in I} \subseteq \mathbb{PA}$ be a chain. Since $\widetilde{\sum}$ is the *lub* operation on $\mathbb{PA}$, we have to prove that $\widetilde{\sum}\{\mathcal{P}_\alpha[\![\mathsf{P}]\!]_{\mathsf{X}_i}\}_{i\in I} = \mathcal{P}_\alpha[\![\mathsf{P}]\!]_{\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I}}$. First of all, note that the following equalities hold.

$$
\begin{array}{ll}
\mathcal{C}_\alpha[\![\mathsf{c}]\!]_{\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I}} = & [\text{by definition of } \mathcal{C}_\alpha[\![\cdot]\!]\,] \\
\alpha(\mathit{tree}(\mathsf{c})) \,\widetilde{\triangleright}\, \mathcal{G}_\alpha[\![\mathbf{B}]\!]_{\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I}} = & [\text{by Point 2 of Theorem 3.6.2}] \\
\alpha(\mathit{tree}(\mathsf{c})) \,\widetilde{\triangleright}\, \widetilde{\sum}\{\mathcal{G}_\alpha[\![\mathbf{B}]\!]_{\mathsf{X}_i}\}_{i\in I} = & [\text{by (3.6.1)}] \\
\widetilde{\sum}\{\alpha(\mathit{tree}(\mathsf{c})) \,\widetilde{\triangleright}\, \mathcal{G}_\alpha[\![\mathbf{B}]\!]_{\mathsf{X}_i}\}_{i\in I} = & [\text{by definition of } \mathcal{C}_\alpha[\![\cdot]\!]\,] \\
\widetilde{\sum}\{\mathcal{C}_\alpha[\![\mathsf{c}]\!]_{\mathsf{X}_i}\}_{i\in I}. &
\end{array}
$$

Now the proof is straightforward by definition of $\mathcal{P}_\alpha[\![\cdot]\!]$. ∎

Finally, by using Theorems 3.6.9 and 3.6.6, we can prove the equivalences between the denotational and the operational semantics on one side, and between the top-down and bottom-up denotations on the other side.

Corollary 3.6.10 *Let* $\alpha : \mathbb{C} \to \mathbb{A}$ *be a semi-perfect observable,* $\mathbf{G}$ *be a goal and* P, P' *be programs. Then*

1. $\mathcal{O}_\alpha[\![\mathsf{P}]\!] = \mathcal{F}_\alpha[\![\mathsf{P}]\!]$,
2. $\mathcal{Q}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!] = \mathcal{B}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!]$,

Proof. We prove the points separately.

Point 1 We prove the two inclusions separately

$\leq$ First of all, we prove that, for any $\mathsf{n} \geq 1$, $\mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_\mathsf{n}} \geq \alpha((Id_\mathbb{I} + \gamma\alpha(tree(\mathsf{P}))) \triangleright pu^{\mathsf{n}-1}(Id_\mathbb{I} + \gamma\alpha(tree(\mathsf{P}))))$. The proof is by induction on n. For $\mathsf{n} = 1$ the thesis is straightforward, since

$$
\begin{aligned}
&\mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_1} = \\
&\qquad [\text{by definition of } \mathcal{P}_\alpha[\![\cdot]\!] \text{ and } \cdot{\uparrow_1}] \\
&\alpha(Id_\mathbb{I} + (tree(\mathsf{P}) \triangleright pu(\gamma\alpha(\bot_\mathbb{A})))) \geq \\
&\qquad [\text{since } \triangleright \text{ is extensive}] \\
&\alpha(Id_\mathbb{I} + tree(\mathsf{P})) = \\
&\qquad [\text{by } (3.3.1)] \\
&\alpha(Id_\mathbb{I} + \gamma\alpha(tree(\mathsf{P}))) \\
&\qquad [\text{by definition of } pu^0] \\
&\alpha((Id_\mathbb{I} + \gamma\alpha(tree(\mathsf{P}))) \triangleright pu^0(Id_\mathbb{I} + \gamma\alpha(tree(\mathsf{P})))).
\end{aligned}
$$

For $\mathsf{n} > 1$ the following statements hold.

$$
\begin{aligned}
&\mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_\mathsf{n}} = \\
&\qquad [\text{by definition of } \cdot{\uparrow_\mathsf{n}} \text{ and since } \mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}} \leq \mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_\mathsf{n}}] \\
&\mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}}) \mathbin{\widetilde{+}} \mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}} \geq \\
&\qquad [\text{by Point 2 of Theorem 3.6.9 and } (2.5.8)] \\
&\alpha(Id_\mathbb{I} + (tree(\mathsf{P}) \triangleright pu(\gamma(\mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}})))) \mathbin{\widetilde{+}} \mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}} = \\
&\qquad [\text{by } (3.3.1) \text{ and Point 4 of Theorem 3.6.2}] \\
&\alpha(Id_\mathbb{I} + \gamma\alpha(\gamma\alpha(tree(\mathsf{P})) \triangleright pu(\gamma(\mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}})))) \mathbin{\widetilde{+}} \mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}} = \\
&\qquad [\text{ by definition of } \widetilde{+}\ ] \\
&\alpha(\gamma\alpha(tree(\mathsf{P})) \triangleright pu(\gamma(\mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}}))) \mathbin{\widetilde{+}} (\alpha(Id_\mathbb{I}) \mathbin{\widetilde{+}} \mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}}) \geq \\
&\qquad [\text{since } \gamma\alpha \text{ is extensive and by definition of } \widetilde{+}] \\
&\alpha((\gamma\alpha(tree(\mathsf{P})) \triangleright pu(\gamma(\mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}}))) + (Id_\mathbb{I} + \gamma(\mathcal{P}_\alpha[\![\mathsf{P}]\!]{\uparrow_{\mathsf{n}-1}}))) = \\
&\qquad [\text{since } Id_\mathbb{I} + \mathsf{D} = Id_\mathbb{I} \triangleright pu(\mathsf{D}), \text{ for any } \mathsf{D} \in \mathbb{PC}]
\end{aligned}
$$

$$\alpha((\gamma\alpha(tree(\mathsf{P})) \triangleright pu(\gamma(\mathcal{P}_\alpha[\![\mathsf{P}]\!]\uparrow_{n-1}))) + (Id_{\mathbb{I}} \triangleright pu(\gamma(\mathcal{P}_\alpha[\![\mathsf{P}]\!]\uparrow_{n-1})))) =$$

[by Lemma 2.2.7]

$$\alpha((Id_{\mathbb{I}} + \gamma\alpha(tree(\mathsf{P}))) \triangleright pu(\gamma(\mathcal{P}_\alpha[\![\mathsf{P}]\!]\uparrow_{n-1}))) =$$

[by inductive hypothesis]

$$\alpha((Id_{\mathbb{I}} + \gamma\alpha(tree(\mathsf{P}))) \triangleright pu(\gamma\alpha((Id_{\mathbb{I}} + \gamma\alpha(tree(\mathsf{P}))) \triangleright pu^{n-2}(Id_{\mathbb{I}} + \gamma\alpha(tree(\mathsf{P})))))) =$$

[since $\gamma\alpha$ is extensive and by definition of pu^{n-1}]

$$\alpha((Id_{\mathbb{I}} + \gamma\alpha(tree(\mathsf{P}))) \triangleright pu^{n-1}(Id_{\mathbb{I}} + \gamma\alpha(tree(\mathsf{P})))).$$

Now, since $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ is continuous on $\mathbb{A}$, by Lemma 2.2.7 and by (3.3.1), $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \widetilde{\sum}\{\mathcal{P}_\alpha[\![\mathsf{P}]\!]\uparrow_n\}_{n\geq 0} \geq \widetilde{\sum}\{\alpha((Id_{\mathbb{I}} + \gamma\alpha(tree(\mathsf{P}))) \triangleright pu^n(Id_{\mathbb{I}} + \gamma\alpha(tree(\mathsf{P}))))\}_{n\geq 0} \geq \alpha(Id_{\mathbb{I}} \triangleright \sum\{pu^n(\gamma\alpha(tree(\mathsf{P})))\}_{n\geq 0})$. Then, by Corollary 2.5.13, since α is monotonic and by Lemma 3.6.5, $\mathcal{F}_\alpha[\![\mathsf{P}]\!] \geq \alpha(Id_{\mathbb{I}} \triangleright \sum\{su_n(\gamma\alpha(tree(\mathsf{P})))\}_{n\geq 0}) = \mathcal{O}_\alpha[\![\mathsf{P}]\!]$.

$\geq$ Since (by Corollary 3.6.3) any semi-perfect observable is also semi-denotational, the proof is the same of Lemma 3.5.5, by using Point 2 of Theorem 3.6.9 instead of Point 1 of Theorem 3.5.2.

Point 2 The proof is straightforward by Point 1 and by Theorem 3.6.6. ■

3.6.1 *SLD*-trees with $\mathcal{POS}$

We show now how to obtain *groundness analysis of SLD-trees* by applying our scheme. We will obtain abstract computations on the abstract domain $\mathcal{POS}$ by extending the Γ abstraction of Section 3.5.1 to derivations. First of all we need to define abstract versions of atoms, goals, clauses and programs.

$$\begin{aligned}
\Gamma(t_1,\ldots,t_n) &:= \Gamma(t_1),\ldots,\Gamma(t_n)\\
\Gamma(p(\mathbf{t})) &:= p(\Gamma(\mathbf{t}))\\
\Gamma(A_1,\ldots,A_n) &:= \Gamma(A_1),\ldots,\Gamma(A_n)\\
\Gamma(H \leftarrow \mathbf{B}) &:= \Gamma(H) \leftarrow \Gamma(\mathbf{B})\\
\Gamma(\{c_1,\ldots,c_n\}) &:= \{\Gamma(c_1),\ldots,\Gamma(c_n)\}
\end{aligned}$$

Now we can extend Γ to derivations. We call $\mathcal{POS}$*-derivation* the result of this abstraction, since it represents an abstract derivation on the domain $\mathcal{POS}$.

$$\Gamma\left(\mathbf{G}_0 \xrightarrow[c_1]{\vartheta_1} \cdots \xrightarrow[c_n]{\vartheta_n} \mathbf{G}_n\right) := \Gamma(\mathbf{G}_0) \xrightarrow[\Gamma(c_1)]{\Gamma(\vartheta_1)} \cdots \xrightarrow[\Gamma(c_n)]{\Gamma(\vartheta_n)} \Gamma(\mathbf{G}_n)$$

Let WFS_{POS} be the set of well formed sets of $\mathcal{POS}$-derivations and $\mathbb{A}_{\mathrm{SLDPOS}} \subseteq [Goals \rightharpoonup WFS_{\mathrm{POS}}]$.

The *groundness dependencies of SLD-tree* observable $\Psi : \mathbb{C} \to \mathbb{A}_{\mathrm{SLDPOS}}$ is

$$\Psi(\mathsf{D}) := \lambda \mathbf{G}. \left\{ \Gamma(\mathsf{d}) \mid \mathsf{d} \in \mathsf{D}(\mathbf{G}) \right\}.$$

It can be proved that the observable Ψ is a semi-perfect observable, hence the abstract transition relation $\xrightarrow[\mathsf{P}]{\Psi}$ is precise. In order to present it in a compact form, we introduce the following notation. Let $\mathsf{A} := \mathsf{p}(\mathsf{F}_1, \ldots, \mathsf{F}_n)$ and $\mathsf{B} := \mathsf{q}(\mathsf{E}_1, \ldots, \mathsf{E}_m)$ be abstract atoms. If $\mathsf{p}/n = \mathsf{q}/m$, then $\mathsf{A} \leftrightarrow \mathsf{B}$ denotes $\bigwedge_{1 \leq i \leq n} \mathsf{F}_i \leftrightarrow \mathsf{E}_i$, otherwise it denotes *false*. Thus, by applying the definition the abstract transition system, if for some $\mathsf{X} \in \mathbb{A}_{\mathrm{SLDPOS}}$ there exists an abstract goal $\bar{\mathbf{G}}$, an abstract derivation $\mathsf{A} \xrightarrow[c_1]{F_1} \cdots \xrightarrow[c_n]{F_n} (\mathsf{B}, \mathbf{C}) \in \mathsf{X}(\bar{\mathbf{G}})$ and an abstract clause $\mathsf{H} \leftarrow \mathbf{D} \in \Gamma(\mathsf{P})$ such that $(\mathsf{B} \leftrightarrow \mathsf{H}) \neq \mathit{false}$, then we can do the derivation step

$$\begin{aligned} \mathsf{X} \xrightarrow[\mathsf{P}]{\Psi} \lambda \mathbf{G}. \mathsf{X}(\mathbf{G}) \cup \Big\{ & \mathsf{A} \xrightarrow[c_1]{F_1} \cdots \xrightarrow[c_n]{F_n} (\mathsf{B}, \mathbf{C}) \xrightarrow[\mathsf{H} \leftarrow \mathbf{D}]{F_n \wedge \mathsf{B} \leftrightarrow \mathsf{H}} (\mathbf{D}, \mathbf{C}) \mid \mathsf{H} \leftarrow \mathbf{D} \in \Gamma(\mathsf{P}), \\ & \mathsf{A} \xrightarrow[c_1]{F_1} \cdots \xrightarrow[c_n]{F_n} (\mathsf{B}, \mathbf{C}) \in \mathsf{X}(\mathbf{G}), (\mathsf{B} \leftrightarrow \mathsf{H}) \neq \mathit{false} \Big\} \end{aligned}$$

Since Ψ is semi-perfect, the abstract top-down denotation $\mathcal{O}_\Psi[\![\mathsf{P}]\!]$ and the abstract bottom-up denotation $\mathcal{F}_\Psi[\![\mathsf{P}]\!]$ coincide and are less precise the abstraction of the top-down denotation $\Psi(\mathcal{O}[\![\mathsf{P}]\!])$.

Let us finally note that we can obtain the observable ψ (see Section 3.5.1) by collecting the results of $\mathcal{POS}$-refutations provided by Ψ. Namely, $\psi = \pi \circ \Psi$, where

$$\pi(\mathsf{X}) := \lambda \mathbf{G}. \left\{ (\mathsf{F}_1 \wedge \cdots \wedge \mathsf{F}_n)|_{\mathsf{G}} \mid \Gamma(\mathbf{G}) \xrightarrow[c_1]{F_1} \cdots \xrightarrow[c_n]{F_n} \square \in \mathsf{X}(\mathbf{G}) \right\}.$$

Thus, since ψ is semi-denotational and its transition system is not precise, we can use that of Ψ and then collect abstract computer answers.

3.7 Discussion on the Results of the Abstraction Framework

Our results allow us to derive in a systematic way several known semantics, together with their properties. Perfect observables allow us to reconstruct the resultants semantics in [54] and the Heyting semantics in [67]. The results on denotational observables apply to the least Herbrand model, the atomic logical consequence semantics [16], the s-semantics [45, 13], the partial answer semantics [43] and the call pattern semantics [55]. The results on semi-denotational observables apply to

bottom-up abstract interpretation frameworks, such as those whose collecting semantics is the computed answer semantics [9, 17], the correct answer semantics [59] and the call pattern semantics [51]. Finally, the results on semi-perfect observables apply to top-down abstract interpretation frameworks [15].

There is something more we can learn from our theory of observables on the relation between denotational and operational definitions and the relation between *goal-independence* and *goal-dependence*. Denotational definitions are usually more abstract than operational definitions. This is not a real need, since we have shown that denotational definitions do exist for all sensible observables (see the discussion below on operational observables). In particular, for those observables which have a precise operational semantics (i.e., perfect observables, such as *SLD*-trees, resultants and proof trees), there exists a fully equivalent denotational definition. The difference between an operational and denotational definition is not a matter of level of abstraction of the semantic domains (which can be "very operational" as in the case of *SLD*-trees). It is rather a matter of style, namely the denotational definition is compositional (and, in the case of procedure definition and procedure call, this leads to goal-independence).

However, it is often the case that a denotational definition is compared to an operational definition modeling a different observable. For example, when one is concerned with computed answers (or their ground instances),

- the operational semantics is usually defined by *SLD*-resolution, i.e., by a transition system for the observable *SLD*-trees. Computed answers (or their ground instances) are then obtained by abstracting the final result;

- the denotational semantics is defined directly on the abstract domain by taking the least fixpoint of the s-semantics immediate consequence operator (or of the standard ground T_P operator).

This can easily be explained in our framework. In fact, computed answers and their ground instances are denotational, and can therefore be computed operationally only by taking a more concrete (perfect) observable. As a consequence, the operational semantics is "more expressive" and yet contains more information. For example, from this operational semantics designed for computed answers, we can obtain information about call patterns, since both computed answers and call patterns are abstractions of the observable we are indeed modeling.

It is worth noting that in principle there might exist observables which can only be computed operationally. We just need to define another class, which satisfies axioms (3.4.1) and (3.4.2) and an axiom on the extension operator symmetric w.r.t.

Axiom (3.4.3), i.e.,

$$\alpha(\mathsf{D} \triangleright \mathsf{D}') = \alpha((\gamma \circ \alpha)\mathsf{D} \triangleright \mathsf{D}').$$

This class (operational observables) has been studied in [2] and seems to include no interesting observables.

The relation between operational and denotational definitions that we have noticed for precise observables, holds for non-precise observables too. Consider now the case of groundness analysis, based on the abstract domain $\mathcal{POS}$. As in the case of precise observables, the operational and denotational definitions, use different observables. Namely,

- the (top-down) abstract operational semantics models a semi-perfect observable (the one of Section 3.6.1). Observations (i.e., groundness dependencies for computed answers) are then obtained by abstracting the result;

- the (bottom-up) abstract denotational semantics models directly a semi-denotational observable (the one of Section 3.5.1).

As in the case of precise observables, the two definitions are equivalent from the viewpoint of precision. However, since the top-down abstract semantics is more concrete, it allows to derive more information (e.g., groundness dependencies of call patterns).

It is worth noting that all the observables we have considered are *AND*-compositional. *AND*-compositionality implies *goal-independence*, i.e., there is no loss of precision in deriving the behavior of a specific goal from the goal-independent denotation (this property is sometimes called "condensing" in abstract interpretation). However there exist abstract domains used in the static analysis of logic programs, for which this property does not hold. One example is the domain $\mathcal{DEF}$ [8], which is a domain of propositional formulas (less precise than $\mathcal{POS}$) used for groundness analysis. It is easy to realize that $\mathcal{DEF}$ is not semi-denotational since it does not satisfy Axiom (3.5.1). In the case of non-*AND*-compositional observables, the denotational definition (which is *AND*-compositional by construction) will in general deliver results which are less precise than those that could be obtained by goal-dependent operational definitions (defined for the "corresponding" more concrete observable). This is the only case where goal-dependent top-down abstract interpretation is more precise than goal-independent bottom-up abstract interpretation.

Our final remark is about goal-independence. Goal-dependence is usually associated to top-down operational definitions. One example is *SLD*-resolution, which given a goal, returns the meaning of the goal. On the other side, goal-independence

is usually associated to bottom-up denotational definitions. For example, the fixpoint semantics gives a meaning to a set of procedure declarations. Our framework shows that this is not always the case. In particular, one can get the meaning of a goal in a denotational way (see definitions (2.2.1) and (3.2.8) for the four classes of observables we have considered). However, the meaning of a goal is compositionally derived from the (goal-independent) meaning of the clauses. As already noted, if the observable is *AND*-compositional, the denotational semantics of a goal is precise. On the other hand, goal-independent denotations can be defined also in terms of the operational semantics, if the observable is perfect or semi-perfect, by taking the behaviors for pure atomic goals. This shows that top-down does not necessarily imply goal-dependence and bottom-up does not necessarily imply goal-independence.

3.8 Applications and Future Developments

Our framework was explicitly defined as a tool for the reconstruction of existing semantics, for the systematic design of new semantics and for static program analysis. One additional application is *abstract diagnosis* [26, 24, 23], a generalization of declarative debugging, which allows us to prove whether a program satisfies an abstract specification and to locate the bugs, when the program is not correct. Abstract diagnosis is extensively presented in Part II. Let us just note now, that the diagnosis algorithms explicitly exploit the properties of denotational and semi-denotational observables.

Another application which is currently under study is the *systematic* design of observables with specific semantic properties. The problem can be stated as follows. We want to model the observable α, by a semantics which has some property (such as being precise, *AND*-compositional, *OR*-compositional, goal-independent, or top-down). If α belongs to a class which does not enjoy that property, we need to determine another observable β which is more concrete than α (and therefore is correct w.r.t. it) and which has the required property.

Consider, for example, the case where one wants to model computed answers by an *OR*-compositional semantics (because one needs to reason in a modular way). Computed answers are denotational and not perfect and therefore they are not *OR*-compositional. An *OR*-compositional semantics, correct w.r.t. computed answers, was defined in [14]. It turns out to be exactly the semantics for the observable computed resultants, which is indeed perfect and therefore *OR*-compositional.

As another example, assume one wants to model groundness dependencies of computed answers by an *AND*-compositional (goal-independent) semantics. The simple "groundness" observable is not even semi-denotational. On the other hand,

$\mathcal{POS}$ is correct w.r.t. groundness dependencies of computed answers and is semi-denotational (and therefore *AND*-compositional).

The theory of abstract interpretation provides tools for the systematic construction of "more concrete" observables, i.e., *refinement operators* (see [50] for a recent survey). Examples of refinement operators are reduced product, disjunctive completion, functional dependencies and Heyting completion. The above mentioned examples can be handled by these techniques. In particular [60] shows that the resultants semantics can be obtained by refining the domain of the s-semantics by functional dependencies, while [62, 89] reconstruct $\mathcal{POS}$ from the "groundness" observable by Heyting completion. Within our framework, one can handle the problem of establishing general results about the properties of a class of observables and those of their refinements. Initial results for precise observables can be found in [2].

As a final remark, we want to point out that our approach can be generalized to other paradigms. We just need to define a denotational and operational semantics on the same semantic domain. The compositionality properties will be of course different and related to the language syntactic operators.

3.9 Proofs of Examples

We present here the proofs of the properties stated in the examples. We first need some technical results on properties of resultants and derivations. To simplify the notation, by $res(\mathtt{d}) := \mathbf{G}_0\vartheta \leftarrow \mathbf{G}_n$ we will denote the resultant associated to the derivation $\mathtt{d}$.

Lemma 3.9.1 *Let* $\mathtt{d} := \mathbf{G} \xrightarrow[c_1,\ldots,c_m]{\vartheta'} \mathbf{G}'_m$ *be a derivation and* δ *be an idempotent substitution such that* $var(\mathbf{G}\delta) \cap var(clauses(\mathtt{d})) = \emptyset$. *Then there exists* ϑ *such that* $\partial_\delta(\mathtt{d}) = \mathbf{G}\delta \xrightarrow[c_1,\ldots,c_m]{\vartheta} \mathbf{G}_m$, *if and only if there exists* $\delta' = mgu(\mathbf{G}\delta, \mathbf{G}\vartheta')$, *such that* $res(\partial_\delta(\mathtt{d})) = res(\mathtt{d})\delta'$.

Proof. First of all observe that by definition of derivation and of the ∂ operation, $var(clauses(\mathtt{d})) \cap (var(\mathbf{G}\delta) \cup var(\mathbf{G})) = \emptyset$ and δ is an idempotent substitution. Then, by Lemma 1.3.8 and by observing that $\vartheta'\delta' = \delta|_{\mathbf{G}}\vartheta$, $\mathbf{G}\delta\vartheta \leftarrow \mathbf{G}_m = (\mathbf{G}\vartheta' \leftarrow \mathbf{G}'_m)\delta'$ and therefore $res(\partial_\delta(\mathtt{d})) = res(\mathtt{d})\delta'$. ■

Lemma 3.9.2 *Let* $\mathtt{d}_1 := \mathbf{G}_1 \xrightarrow[c_1,\ldots,c_n]{\vartheta} \mathbf{B}_1$ *and* $\mathtt{d}_2 := \mathbf{G}_2 \xrightarrow[c_{n+1},\ldots,c_{n+m}]{\sigma} \mathbf{B}_2$ *be derivations such that* $var(\mathtt{d}_1) \cap var(\mathtt{d}_2) = var(\mathbf{G}_1) \cap var(\mathbf{G}_2)$. *Then the following facts hold*

1. *If* $\mathbf{B}_1 \neq \square$ *then* $\mathtt{d}_1 \wedge \mathtt{d}_2 = (\mathbf{G}_1, \mathbf{G}_2) \xrightarrow[c_1,\ldots,c_n]{\vartheta} (\mathbf{B}_1, \mathbf{G}_2\vartheta)$ *and* $res(\mathtt{d}_1 \wedge \mathtt{d}_2) = ((\mathbf{G}_1, \mathbf{G}_2) \leftarrow (\mathbf{B}_1, \mathbf{G}_2))\vartheta$.

2. *If* $\mathbf{B}_1 = \square$ *then there exists* ϑ' *such that* $\mathtt{d}_1 \wedge \mathtt{d}_2 = (\mathbf{G}_1, \mathbf{G}_2) \xrightarrow[c_1,\ldots,c_{n+m}]{\vartheta\vartheta'} \mathbf{B}$ *if and only if there exists* $\delta' = mgu(\mathbf{G}_2\vartheta, \mathbf{G}_2\sigma)$ *such that* $res(\mathtt{d}_1 \wedge \mathtt{d}_2) = ((\mathbf{G}_1, \mathbf{G}_2) \leftarrow \mathbf{B}_2)\vartheta\delta'$.

Proof. We prove the points separately.

Point 1 The proof follows by definition of ∂, by a straightforward inductive argument and by observing that (since we use only idempotent mgus) $\mathbf{B}_1\vartheta = \mathbf{B}_1$.

Point 2 By definition of $\wedge$, $\mathtt{d}_1 \wedge \mathtt{d}_2 = \mathtt{d}_1 \wedge \mathbf{G}_2 :: \partial_\vartheta(\mathtt{d}_2)$, where $\mathtt{d}_1 \wedge \mathbf{G}_2 = (\mathbf{G}_1, \mathbf{G}_2) \xrightarrow[c_1,\ldots,c_n]{\vartheta} \mathbf{G}_2\vartheta$. By Lemma 3.9.1 there exists ϑ' such that

$$\partial_\vartheta(\mathtt{d}_2) = \mathbf{G}_2\vartheta \xrightarrow[c_{n+1},\ldots,c_{n+m}]{\vartheta'} \mathbf{B}$$

if and only if there exists $\delta' = mgu(\mathbf{G}_2\vartheta, \mathbf{G}_2\sigma)$, such that

$$\mathbf{G}_2\vartheta\vartheta' \leftarrow \mathbf{B} = (\mathbf{G}_2\sigma \leftarrow \mathbf{B}_2)\delta' = (\mathbf{G}_2\vartheta \leftarrow \mathbf{B}_2)\delta',$$

where the last equality follows since $\delta' = mgu(\mathbf{G}_2\vartheta, \mathbf{G}_2\sigma)$. Since $res(\mathtt{d}_1 \wedge \mathtt{d}_2) = (\mathbf{G}_1, \mathbf{G}_2)\vartheta\vartheta' \leftarrow \mathbf{B}$ we are left to prove the following.

$\mathbf{G}_1\vartheta\vartheta' = \mathbf{G}_1\vartheta\delta'$ It suffices to prove that, for any $x \in var(\mathbf{G}_1)$, $x\vartheta\vartheta' = x\vartheta\delta'$. Let $y \in var(x\vartheta)$. We have two possibilities.

$y \in var(\mathbf{G}_2\vartheta)$ In this case, since $\mathbf{G}_2\vartheta\vartheta' = \mathbf{G}_2\vartheta\delta'$, $y\vartheta' = y\delta'$.

$y \notin var(\mathbf{G}_2\vartheta)$ First of all observe that, since ϑ is idempotent and $y \in var(x\vartheta)$, $y \notin dom(\vartheta)$ and therefore $y \notin var(\mathbf{G}_2)$, since $y \notin var(\mathbf{G}_2\vartheta)$. Moreover, by hypothesis, $var(\mathtt{d}_1) \cap var(clauses(\mathtt{d}_2)) = \emptyset$. Then, by definition of derivation, $y \notin var(\sigma)$ (since $y \notin var(\mathbf{G}_2)$) and $y \notin var(\vartheta')$ (since $y \notin var(\mathbf{G}_2\vartheta)$). Then, since $\delta' = mgu(\mathbf{G}_2\vartheta, \mathbf{G}_2\sigma)$ and (by the previous observations) $y \notin var(\mathbf{G}_2\vartheta) \cup var(\mathbf{G}_2\sigma)$, $y \notin var(\delta')$. Then $y\vartheta' = y = y\delta'$.

$\mathbf{B}_2\vartheta\delta' = \mathbf{B}_2\delta'$ First of all recall that in a derivation only idempotent *mgus* and renamed apart clauses are used. Then it is easy to check, by a straightforward inductive argument, that σ is idempotent and $\mathbf{B}_2\sigma = \mathbf{B}_2$. It suffices to prove that, for any $x \in var(\mathbf{B}_2)$, $x\vartheta\delta' = x\delta'$. We have two cases.

$x \in var(\mathbf{G}_2)$ In this case, since $\delta' = mgu(\mathbf{G}_2\vartheta, \mathbf{G}_2\sigma)$ and by the previous observation, we have that $x\vartheta\delta' = x\sigma\delta' = x\delta'$.

$\underline{\boldsymbol{x} \notin \boldsymbol{var}(\mathbf{G}_2)}\rfloor$ In this case, since by hypothesis $var(\mathtt{d}_1) \cap var(\mathtt{d}_2) \subseteq var(\mathbf{G}_2)$ and $var(\vartheta) \subseteq var(\mathtt{d}_1)$, we have that $\mathtt{x} \notin dom(\vartheta)$ and therefore $\mathtt{x}\vartheta\delta' = \mathtt{x}\delta'$.

■

Now we can prove the properties stated in the examples.

Proof of Example 3.1.3. By definition, ξ maps finite elements to finite elements and $\langle \xi, \xi^{\gamma} \rangle$ satisfies Points 1 and 2 of Definition 3.1.1. We have to prove Point 3 of Definition 3.1.1.

Let $\mathsf{D}, \mathsf{D}' \in \mathbb{PC}$ and $\mathsf{D} \equiv_{\mathbb{C}} \mathsf{D}'$. First of all note that by definition of ξ and since D is a pure collection, $\xi^{\gamma}\xi(\mathsf{D})$ is also a pure collection. Then we have only to prove that, for any goal $\mathsf{p}(\boldsymbol{x})$ such that $\xi^{\gamma}\xi(\mathsf{D})(\mathsf{p}(\boldsymbol{x}))$ is defined, there exists a renaming ρ such that $\xi^{\gamma}\xi(\mathsf{D}')(\mathsf{p}(\boldsymbol{x})\rho)$ is defined and, for any $\mathtt{d} \in \xi^{\gamma}\xi(\mathsf{D})(\mathsf{p}(\boldsymbol{x}))$, there exists $\mathtt{d}' \in \xi^{\gamma}\xi(\mathsf{D}')(\mathsf{p}(\boldsymbol{x})\rho)$ such that $clauses(\mathtt{d}) \equiv clauses(\mathtt{d}')$. Then the thesis follows by symmetry.

Assume that $\xi^{\gamma}\xi(\mathsf{D})(\mathsf{p}(\boldsymbol{x}))$ is defined. By definition of ξ, $\mathsf{D}(\mathsf{p}(\boldsymbol{x}))$ is also defined. Then, since (by hypothesis) $\mathsf{D} \equiv_{\mathbb{C}} \mathsf{D}'$, there exists a renaming ρ such that $\mathsf{D}'(\mathsf{p}(\boldsymbol{x})\rho)$ is defined and for any $\bar{\mathtt{d}} \in \mathsf{D}(\mathsf{p}(\boldsymbol{x}))$ there exists $\bar{\mathtt{d}}' \in \mathsf{D}'(\mathsf{p}(\boldsymbol{x})\rho)$ such that $clauses(\bar{\mathtt{d}}) \equiv clauses(\bar{\mathtt{d}}')$. Moreover, by definition of ξ and since $\mathsf{D}'(\mathsf{p}(\boldsymbol{x})\rho)$ is defined, $\xi^{\gamma}\xi(\mathsf{D}')(\mathsf{p}(\boldsymbol{x})\rho)$ is also defined. Now let $\mathtt{d} \in \xi^{\gamma}\xi(\mathsf{D})(\mathsf{p}(\boldsymbol{x}))$. We are left to prove that there exists $\mathtt{d}' \in \xi^{\gamma}\xi(\mathsf{D}')(\mathsf{p}(\boldsymbol{x})\rho)$ such that $clauses(\mathtt{d}) \equiv clauses(\mathtt{d}')$. Two cases arise.

$\underline{\boldsymbol{last}(\mathtt{d}) \neq \square}\rfloor$ Note that $last(\mathtt{d}\rho) \neq \square$. Then, by definition of ξ and since $\mathsf{D}'(\mathsf{p}(\boldsymbol{x})\rho)$ is defined, $\mathtt{d}\rho \in \xi^{\gamma}\xi(\mathsf{D}')(\mathsf{p}(\boldsymbol{x})\rho)$ and, by definition of renaming, $clauses(\mathtt{d}) \equiv clauses(\mathtt{d}\rho)$.

$\underline{\boldsymbol{last}(\mathtt{d}) = \square}\rfloor$ By definition of ξ, there exists $\bar{\mathtt{d}} \in \mathsf{D}(\mathsf{p}(\boldsymbol{x}))$ such that $answer(\mathtt{d}) = answer(\bar{\mathtt{d}})$. Then, since (by hypothesis) $\mathsf{D} \equiv_{\mathbb{C}} \mathsf{D}'$, there exists $\bar{\mathtt{d}}' \in \mathsf{D}'(\mathsf{p}(\boldsymbol{x})\rho)$ such that $clauses(\bar{\mathtt{d}}) \equiv clauses(\bar{\mathtt{d}}')$. By Lemma 1.3.15 there exists an equational derivation $\mathsf{p}(\boldsymbol{x}) \xrightarrow[\bar{c}_1,\ldots,\bar{c}_n]{\text{eq}} \bar{\mathsf{E}}, \square$ corresponding to $\bar{\mathtt{d}}$ such that $\bar{\vartheta} = mgu(\bar{\mathsf{E}})$ and $answer(\mathtt{d}) = answer(\bar{\mathtt{d}}) = \bar{\vartheta}|_{\boldsymbol{x}}$. Moreover, by Lemma 1.3.15 again, there exists an equational derivation $\mathsf{p}(\boldsymbol{x})\rho \xrightarrow[\bar{c}_1\rho,\ldots,\bar{c}_n\rho]{\text{eq}} \bar{\mathsf{E}}\rho, \square$ corresponding to $\bar{\mathtt{d}}'$ such that $\bar{\vartheta}' = mgu(\bar{\mathsf{E}}\rho)$ and $answer(\bar{\mathtt{d}}') = \bar{\vartheta}'|_{\boldsymbol{x}\rho}$.

Since $\boldsymbol{x}$ is a sequence of distinct variables, there exists a solved form of $\bar{\mathsf{E}}$, $\bar{\mathsf{E}}|_{\boldsymbol{x}} \cup \bar{\mathsf{E}}|_{-\boldsymbol{x}}$, where $\bar{\mathsf{E}}|_{\boldsymbol{x}} \approx_e \{\boldsymbol{x} =_e \mathbf{t}\}$ and by Lemma 1.3.3 there exist $\bar{\delta} = mgu(\bar{\mathsf{E}}|_{\boldsymbol{x}})$ and $\bar{\sigma} = mgu(\bar{\mathsf{E}}|_{-\boldsymbol{x}}\bar{\delta})$ such that $\bar{\vartheta} = \bar{\delta}\bar{\sigma}$. Moreover, since $\bar{\mathsf{E}}|_{\boldsymbol{x}} \cup \bar{\mathsf{E}}|_{-\boldsymbol{x}}$ is a solved form, we have that $\bar{\mathsf{E}}|_{-\boldsymbol{x}}\bar{\delta} = \bar{\mathsf{E}}|_{-\boldsymbol{x}}$, $\{\boldsymbol{x}\} \subseteq var(\bar{\delta})$ and $dom(\bar{\sigma}) \cap var(\bar{\delta}) = \emptyset$. Therefore $\bar{\vartheta}|_{\boldsymbol{x}} = \bar{\delta}|_{\boldsymbol{x}}$.

Analogously, there exists a solved form of $\bar{\mathsf{E}}\rho$, $\bar{\mathsf{E}}\rho|_{\mathbf{x}\rho} \cup \bar{\mathsf{E}}\rho|_{-\mathbf{x}\rho}$, where $\bar{\mathsf{E}}\rho|_{\mathbf{x}\rho} \approx_e \{\mathbf{x}\rho =_e \mathbf{t}\rho\}$ and by Lemma 1.3.3 there exist $\bar{\delta}' = mgu(\bar{\mathsf{E}}\rho|_{\mathbf{x}\rho})$ and $\bar{\sigma}' = mgu(\bar{\mathsf{E}}\rho|_{-\mathbf{x}\rho}\bar{\delta}')$ such that $\bar{\vartheta}' = \bar{\delta}'\bar{\sigma}'$, $\bar{\mathsf{E}}\rho|_{-\mathbf{x}\rho}\bar{\delta}' = \bar{\mathsf{E}}\rho|_{-\mathbf{x}\rho}$, $\{\mathbf{x}\rho\} \subseteq var(\bar{\delta}')$, $dom(\bar{\sigma}') \cap var(\bar{\delta}') = \emptyset$ and, therefore,

$$answer(\bar{\mathsf{d}}') = \bar{\vartheta}'|_{\mathbf{x}\rho} = \bar{\delta}'|_{\mathbf{x}\rho}. \tag{1}$$

Now, since $\mathsf{d} \in \xi^{\gamma}\xi(\mathsf{D})(\mathsf{p}(\mathbf{x}))$, by Lemma 1.3.15 there exists an equational derivation $\mathsf{p}(\mathbf{x}) \xrightarrow[c_1,\ldots,c_k]{eq} \mathsf{E}, \Box$ corresponding to d such that $\vartheta = mgu(\mathsf{E})$ and, since $answer(\mathsf{d}) = answer(\bar{\mathsf{d}})$, $\vartheta|_{\mathbf{x}} = \bar{\vartheta}|_{\mathbf{x}}$.

Then, analogously to the previous cases, since $\mathbf{x}$ is a sequence of distinct variables, there exists a solved form of E, $\mathsf{E}|_{\mathbf{x}} \cup \mathsf{E}|_{-\mathbf{x}}$, where $\mathsf{E}|_{\mathbf{x}} \approx_e \{\mathbf{x} =_e \mathbf{t}'\}$ and $\mathbf{t}'$ is a renamed version of $\mathbf{t}$. Moreover, since the clauses are renamed apart, there exists a renaming ρ' such that $\mathbf{t}'\rho\rho' = \mathbf{t}\rho$ and $\mathbf{x}\rho\rho' = \mathbf{x}\rho$.

Then, by definition of equational derivation, it is easy to check that there exists an equational derivation

$$\mathsf{p}(\mathbf{x})\rho \xrightarrow[c_1\rho\rho',\ldots,c_k\rho\rho']{eq} \mathsf{E}\rho\rho', \Box \tag{2}$$

and, analogously to the previous cases, there exists a solved form $\bar{\mathsf{E}}\rho|_{\mathbf{x}\rho} \cup \widetilde{\mathsf{E}}$ such that $\bar{\delta}' = mgu(\bar{\mathsf{E}}\rho|_{\mathbf{x}\rho})$, $\sigma' = mgu(\widetilde{\mathsf{E}})$, $\vartheta' = mgu(\mathsf{E}\rho\rho') = \bar{\delta}'\sigma'$, $\{\mathbf{x}\rho\} \subseteq var(\bar{\delta}')$, $dom(\sigma') \cap var(\bar{\delta}') = \emptyset$ and, therefore,

$$\vartheta'|_{\mathbf{x}\rho} = \bar{\delta}'|_{\mathbf{x}\rho}. \tag{3}$$

By Lemma 1.3.15 and by (2), there exists a derivation

$$\mathsf{d}' = \mathsf{p}(\mathbf{x}\rho) \xrightarrow[c_1\rho\rho',\ldots,c_k\rho\rho']{\vartheta'} \Box. \tag{4}$$

Moreover, since $\bar{\mathsf{d}}' \in \mathsf{D}'(\mathsf{p}(\mathbf{x})\rho)$, by (1) and (3) and by definition of ξ, $\mathsf{d}' \in \xi^{\gamma}\xi(\mathsf{D}')(\mathsf{p}(\mathbf{x})\rho)$. Finally, by construction, $clauses(\mathsf{d}) \equiv clauses(\mathsf{d}')$ and then the thesis. ■

Proof of Section 3.3.1. We prove that the three abstract operations are correctly defined.

$\tilde{\cdot}$ operation By definition of ∂, given a derivation d and an idempotent substitution δ such that $\partial_\delta(\mathsf{d})$ is defined, $length(\partial_\delta(\mathsf{d})) \leq length(\mathsf{d})$. Then there exists a derivation d' which is a prefix of d such that $\partial_\delta(\mathsf{d}) = \partial_\delta(\mathsf{d}')$ and $length(\partial_\delta(\mathsf{d}')) = length(\mathsf{d}')$.

Now observe that by definition of collection, for any $\mathsf{D} \in \mathbb{C}$ and for any $\mathbf{G} \in \mathit{Goals}$, if $\mathsf{D}(\mathbf{G})$ is defined, then $\mathsf{D}(\mathbf{G})$ is a well-formed set of derivations and therefore, for any $\mathtt{d} \in \mathsf{D}(\mathbf{G})$, if $\mathtt{d}'$ is a prefix of $\mathtt{d}$, then $\mathtt{d}' \in \mathsf{D}(\mathbf{G})$. By the previous observations, given an atom A and a collection D,

$$\begin{aligned}\mathsf{A} \cdot \mathsf{D} = \phi\left[{}^{\mathsf{S}}/_{\mathsf{A}}\right] \text{ where } \mathsf{S} = \{&\partial_\delta(\mathtt{d}') \mid \mathsf{S}' \text{ is renamed apart (from } \mathsf{A}) \\ &\text{version of } \mathsf{D}(\mathsf{A}'), \text{ for some } \mathsf{A}' \leq \mathsf{A},\ \mathtt{d}' \in \mathsf{S}', \text{ there exists } \delta \text{ s.t.} \\ &\mathsf{A} = \mathit{first}(\mathtt{d}')\delta,\ \partial_\delta(\mathtt{d}') \text{ is defined and } \mathit{length}(\partial_\delta(\mathtt{d}')) = \mathit{length}(\mathtt{d}')\}.\end{aligned}$$

Then, by Lemma 3.9.1, there exists $\mathtt{d} \in (\mathsf{A} \cdot \mathsf{D})(\mathsf{A})$ if and only if $\mathtt{d} = \partial_\delta(\mathtt{d}')$, S' is renamed apart (from A) version of $\mathsf{D}(\mathsf{A}')$, for some $\mathsf{A}' \leq \mathsf{A}$, $\mathtt{d}' \in \mathsf{S}'$, $\mathsf{H} = \mathit{first}(\mathtt{d}')$, $\mathit{res}(\mathtt{d}') = \mathsf{H}' \leftarrow \mathbf{B}'$, there exists δ such that $\mathsf{A} = \mathsf{H}\delta$, there exists $\vartheta = \mathit{mgu}(\mathsf{A}, \mathsf{H}')$ and $\mathit{res}(\mathtt{d}) = \mathit{res}(\mathtt{d}')\vartheta$. Then the following equivalences hold.

$$\begin{aligned}&\mathsf{A} \tilde{\cdot} \mathsf{X} \\ &= [\text{by definition of } \tilde{\cdot} \text{ and } \chi] \\ &\lambda\mathbf{G}.\{\mathit{res}(\mathtt{d}) \mid \mathtt{d} \in (\mathsf{A} \cdot \chi^{\gamma}(\mathsf{X}))(\mathbf{G})\} \\ &= [\text{by the previous result}] \\ &\phi\left[{}^{\mathsf{R}}/_{\mathsf{A}}\right] \text{ where } \mathsf{R} = \{\ (\mathsf{H}' \leftarrow \mathbf{B}')\vartheta \mid \mathsf{S}' \text{ is renamed apart (from } \mathsf{A}) \\ &\quad \text{version of } \chi^{\gamma}(\mathsf{X})(\mathsf{A}'), \text{ for some } \mathsf{A}' \leq \mathsf{A},\ \mathtt{d}' \in \mathsf{S}',\ \mathsf{H} = \mathit{first}(\mathtt{d}'), \\ &\quad \mathit{res}(\mathtt{d}') = \mathsf{H}' \leftarrow \mathbf{B}' \text{ there exists } \delta \text{ s.t. } \mathsf{A} = \mathsf{H}\delta \text{ and } \vartheta = \mathit{mgu}(\mathsf{A}, \mathsf{H}')\} \\ &= [\text{by definition of } \chi^{\gamma} \text{ and since } \vartheta = \mathit{mgu}(\mathsf{A}, \mathsf{H}')] \\ &\phi\left[{}^{\mathsf{R}}/_{\mathsf{A}}\right] \text{ where } \mathsf{R} = \{(\mathsf{A} \leftarrow \mathbf{B}')\vartheta \mid \langle \mathsf{H}, \mathsf{R}' \rangle \text{ is a renamed apart (from } \mathsf{A}) \\ &\quad \text{version of } \langle \mathsf{A}', \mathsf{X}(\mathsf{A}') \rangle, \text{ for some } \mathsf{A}' \leq \mathsf{A},\ \mathsf{H}' \leftarrow \mathbf{B}' \in \mathsf{R}', \text{ there} \\ &\quad \text{exists } \delta \text{ s.t. } \mathsf{A} = \mathsf{H}\delta \text{ and } \vartheta = \mathit{mgu}(\mathsf{A}, \mathsf{H}')\}.\end{aligned}$$

$\tilde{\times}$ operation By definition of $\wedge$, if $\mathtt{d}_1, \mathtt{d}_2$ are derivations and $\mathtt{d}_1 \wedge \mathtt{d}_2$ is defined then $\mathit{length}(\mathtt{d}_1 \wedge \mathtt{d}_2) \leq \mathit{length}(\mathtt{d}_1) + \mathit{length}(\mathtt{d}_2)$. Therefore, by definition of $\wedge$, there exists $\mathtt{d}_2'$ which is a prefix of $\mathtt{d}_2$ such that $\mathtt{d}_1 \wedge \mathtt{d}_2 = \mathtt{d}_1 \wedge \mathtt{d}_2'$ and $\mathit{length}(\mathtt{d}_1 \wedge \mathtt{d}_2') = \mathit{length}(\mathtt{d}_1) + \mathit{length}(\mathtt{d}_2')$.

Then, analogously to the previous operation, by properties of collections,

$$\begin{aligned}\mathsf{D}_1 \times \mathsf{D}_2 = \lambda\mathbf{G}.\{&\mathtt{d}_1 \wedge \mathtt{d}_2 \mid (\mathbf{G}_1, \mathbf{G}_2) = \mathbf{G} \text{ and for } \mathtt{i} = 1, 2,\ \mathbf{G}_\mathtt{i} \equiv \mathbf{G}_\mathtt{i}',\ \mathtt{d}_\mathtt{i} \\ &\text{is a renamed version of an element in } \mathsf{D}_\mathtt{i}(\mathbf{G}_\mathtt{i}') \text{ s.t.} \\ &\mathbf{G}_\mathtt{i} = \mathit{first}(\mathtt{d}_\mathtt{i}),\ \mathtt{d}_1 \wedge \mathtt{d}_2 \text{ is defined and} \\ &\mathit{length}(\mathtt{d}_1 \wedge \mathtt{d}_2) = \mathit{length}(\mathtt{d}_1) + \mathit{length}(\mathtt{d}_2)\}.\end{aligned}$$

Then, by Lemma 3.9.2, there exists $\mathtt{d} \in (\mathsf{D}_1 \times \mathsf{D}_2)(\mathbf{G})$ if and only if $\mathtt{d} = \mathtt{d}_1 \wedge \mathtt{d}_2$, $(\mathbf{G}_1, \mathbf{G}_2) = \mathbf{G}$ and for $\mathtt{i} = 1, 2$, $\mathbf{G}_\mathtt{i} \equiv \mathbf{G}_\mathtt{i}'$, $\mathtt{d}_\mathtt{i}$ is a renamed version of an element in $\mathsf{D}_\mathtt{i}(\mathbf{G}_\mathtt{i}')$ such that $\mathbf{G}_\mathtt{i} = \mathit{first}(\mathtt{d}_\mathtt{i})$, $\mathbf{B}_\mathtt{i} = \mathit{last}(\mathtt{d}_\mathtt{i})$, $\vartheta_\mathtt{i} = \mathit{answer}(\mathtt{d}_\mathtt{i})$, $\mathit{var}(\mathtt{d}_1) \cap \mathit{var}(\mathtt{d}_2) = \mathit{var}(\mathbf{G}_1) \cap \mathit{var}(\mathbf{G}_2)$ and the following equivalences hold

- either $\mathbf{B}_1 \neq \Box$ and $res(\mathtt{d}) = ((\mathbf{G}_1, \mathbf{G}_2) \leftarrow (\mathbf{B}_1, \mathbf{G}_2))\vartheta_1$ or
- $\mathbf{B}_1 = \Box$ and there exists $\sigma = mgu(\mathbf{G}_2\vartheta_1, \mathbf{G}_2\vartheta_2)$ such that $res(\mathtt{d}) = ((\mathbf{G}_1, \mathbf{G}_2) \leftarrow \mathbf{B}_2)\vartheta_1\sigma$.

Then the following equivalences hold.

$$
\begin{aligned}
&\mathsf{X}_1 \mathbin{\widetilde{\times}} \mathsf{X}_2 \\
&= [\text{by definition of } \widetilde{\times} \text{ and } \chi] \\
&\lambda\mathbf{G}.\{res(\mathtt{d}) \mid \mathtt{d} \in (\chi^\gamma(\mathsf{X}_1) \times \chi^\gamma(\mathsf{X}_2))(\mathbf{G})\} \\
&= [\text{by the previous result}] \\
&\lambda\mathbf{G}.\{((\mathbf{G}_1, \mathbf{G}_2) \leftarrow \mathbf{B})\vartheta \mid \ (\mathbf{G}_1, \mathbf{G}_2) = \mathbf{G},\ \forall i = 1,2,\ \mathbf{G}_i \equiv \mathbf{G}'_i,\ \mathtt{d}_i \text{ is a} \\
&\quad \text{renamed version of an element in } \chi^\gamma(\mathsf{X}_i)(\mathbf{G}'_i) \text{ s.t. } \mathbf{G}_i = first(\mathtt{d}_i), \\
&\quad \vartheta_i = answer(\mathtt{d}_i),\ \mathbf{B}_i = last(\mathtt{d}_i), \\
&\quad var(\mathtt{d}_1) \cap var(\mathtt{d}_2) = var(\mathbf{G}_1) \cap var(\mathbf{G}_2) \text{ and} \\
&\quad \text{if } \mathbf{B}_1 \neq \Box \text{ then } \vartheta = \vartheta_1 \text{ and } \mathbf{B} = (\mathbf{B}_1, \mathbf{G}_2) \\
&\quad \text{else } \vartheta = \vartheta_1 \circ mgu(\mathbf{G}_2\vartheta_1, \mathbf{G}_2\vartheta_2) \text{ and } \mathbf{B} = \mathbf{B}_2\} \\
&= [\text{by definition of } \chi^\gamma] \\
&\lambda\mathbf{G}.\{((\mathbf{G}_1, \mathbf{G}_2) \leftarrow \mathbf{B})\vartheta \mid \mathbf{G} = (\mathbf{G}_1, \mathbf{G}_2),\ \forall i = 1,2,\ \mathbf{G}_i \equiv \mathbf{G}'_i, \\
&\quad \mathtt{r}_i = \mathbf{G}''_i \leftarrow \mathbf{B}_i \text{ is a renamed version of an element in } \mathsf{X}_i(\mathbf{G}'_i), \text{ via a} \\
&\quad \text{renaming } \rho_i \text{ s.t. } \mathbf{G}'_i\rho_i = \mathbf{G}_i, \\
&\quad var(\mathtt{r}_1) \cap var(\mathtt{r}_2) \subseteq var(\mathbf{G}_1) \cap var(\mathbf{G}_2),\ \mathbf{G}_1\vartheta_1 = \mathbf{G}''_1 \text{ and} \\
&\quad \text{if } \mathbf{B}_1 \neq \Box \text{ then } \vartheta = \vartheta_1|_{\mathbf{G}_1},\ \mathbf{B} = (\mathbf{B}_1, \mathbf{G}_2) \\
&\quad \text{else } \mathbf{B} = \mathbf{B}_2,\ \vartheta = \vartheta_1|_{\mathbf{G}_1} \circ mgu(\mathbf{G}_2\vartheta_1|_{\mathbf{G}_1}, \mathbf{G}''_2)\}.
\end{aligned}
$$

$\widetilde{\rhd}$ operation Let $\mathtt{d}_1, \mathtt{d}_2$ be derivations such that $\mathtt{d}_1 :: \mathtt{d}_2$ is defined, $\mathbf{G} := first(\mathtt{d}_1)$, $\mathbf{B} := last(\mathtt{d}_2)$ and (for $i = 1,2$) $\vartheta_i := answer(\mathtt{d}_i)$. By definition of $::$, $res(\mathtt{d}_1 :: \mathtt{d}_2) = \mathbf{G}\vartheta_1\vartheta_2 \leftarrow \mathbf{B}$.

Now observe that, by definition of $::$, all the clauses used in the derivation $\mathtt{d}_2$ are renamed apart w.r.t. $\mathbf{G}$ and $first(\mathtt{d}_2)$. Moreover we assumed that only idempotent *mgus* are used. Then ϑ_1 and ϑ_2 are idempotent substitutions and, by definition of derivation and by a straightforward inductive argument, $\mathbf{B}\vartheta_1\vartheta_2 = \mathbf{B}\vartheta_2 = \mathbf{B}$.

Then, by definition of $\rhd$, there exists $\mathtt{d} \in (\mathsf{D}_1 \rhd \mathsf{D}_2)(\mathbf{G})$ if and only if either $\mathtt{d} \in \mathsf{D}_1(\mathbf{G})$ or $\mathtt{d} = \mathtt{d}_1 :: \mathtt{d}_2$, $\mathtt{d}_1 \in \mathsf{D}_1(\mathbf{G})$, $\mathbf{G}_2 \equiv last(\mathtt{d}_1)$ and $\mathtt{d}_2$ is a renamed version of an element in $\mathsf{D}_2(\mathbf{G}_2)$ such that $last(\mathtt{d}_1) = first(\mathtt{d}_2)$, $var(\mathtt{d}_1) \cap var(\mathtt{d}_2) = var(first(\mathtt{d}_2))$, $\mathbf{B} = last(\mathtt{d}_2)$, for $i = 1,2$, $\vartheta_i = answer(\mathtt{d}_i)$ and $res(\mathtt{d}) = (\mathbf{G} \leftarrow \mathbf{B}_2)\vartheta_1\vartheta_2$. Then the following equivalences hold.

$$
\begin{aligned}
&\mathsf{X}_1 \mathbin{\widetilde{\rhd}} \mathsf{X}_2 \\
&= [\text{by definition of } \widetilde{\rhd} \text{ and } \chi]
\end{aligned}
$$

$\lambda\mathbf{G}.\{res(\mathtt{d}) \mid \mathtt{d} \in (\chi^{\gamma}(\mathsf{X}_1) \triangleright \chi^{\gamma}(\mathsf{X}_2))(\mathbf{G})\}$

$=$ [by the previous result]

$\lambda\mathbf{G}.\{(\mathbf{G} \leftarrow \mathbf{B})\vartheta \mid$ either $\mathtt{d} \in \chi^{\gamma}(\mathsf{X}_1)(\mathbf{G})$, $answer(\mathtt{d}) = \vartheta$ and $last(\mathtt{d}) = \mathbf{B}$ or $\mathtt{d} = \mathtt{d}_1 :: \mathtt{d}_2$, where $\mathtt{d}_1 \in \chi^{\gamma}(\mathsf{X}_1)(\mathbf{G})$, $\mathbf{G}_2 \equiv last(\mathtt{d}_1)$ and $\mathtt{d}_2$ is a renamed version of an element in $\chi^{\gamma}(\mathsf{X}_2)(\mathbf{G}_2)$ s.t. $last(\mathtt{d}_1) = first(\mathtt{d}_2)$, $var(\mathtt{d}_1) \cap var(\mathtt{d}_2) = var(first(\mathtt{d}_2))$, $\vartheta = answer(\mathtt{d}_1) \circ answer(\mathtt{d}_2)$ and $\mathbf{B} = last(\mathtt{d}_2)\}$

$=$ [by definition of χ^{γ}]

$\lambda\mathbf{G}.\mathsf{X}_1(\mathbf{G}) \cup \{(\mathbf{G}' \leftarrow \mathbf{G}_3)\vartheta \mid \mathtt{r}_1 = \mathbf{G}' \leftarrow \mathbf{G}_1 \in \mathsf{X}_1(\mathbf{G})$, $\mathbf{G}_1 \equiv \mathbf{G}_2$, $\mathtt{r}_2 = \mathbf{G}'_2 \leftarrow \mathbf{G}_3$ is a renamed version of an element in $\mathsf{X}_2(\mathbf{G}_2)$, via a renaming ρ s.t. $\mathbf{G}_2\rho = \mathbf{G}_1$, $var(\mathtt{r}_1) \cap var(\mathtt{r}_2) \subseteq var(\mathbf{G}_1)$, $\mathbf{G}_1\vartheta = \mathbf{G}'_2$ and $dom(\vartheta) \subseteq var(\mathbf{G}_1)\}$.

■

We need now a technical result on the substitution abstraction Γ.

Lemma 3.9.3 *Let* $\mathbf{G}$ *be a goal and* ϑ, ϑ' *be idempotent substitutions such that* $\mathbf{G}\vartheta \equiv \mathbf{G}\vartheta'$. *Then* $(\Gamma(\vartheta))|_{var(\mathbf{G})} = (\Gamma(\vartheta'))|_{var(\mathbf{G})}$.

Proof. First of all observe that, for any idempotent substitution ϑ and sequence of variables $\mathbf{x}$, $(\Gamma(\vartheta))|_{\mathbf{x}} = (\Gamma(\vartheta|_{\mathbf{x}}))|_{\mathbf{x}}$. Then we can assume that $(dom(\vartheta) \cup dom(\vartheta')) \subseteq var(\mathbf{G})$.

Now the proof is by induction on $n = card(dom(\vartheta) \setminus dom(\vartheta')) + card(dom(\vartheta') \setminus dom(\vartheta))$.

$\underline{n = 0}$ Assume that $\vartheta \neq \vartheta'$. Then there exists $x \in var(\mathbf{G})$ such that $x\vartheta \neq x\vartheta'$. Since (by hypothesis) $\mathbf{G}\vartheta \equiv \mathbf{G}\vartheta'$, $x\vartheta$ is a variant of $x\vartheta'$ and therefore we can assume that $x\vartheta = f(y, t_2, \ldots, t_k)$, $x\vartheta' = f(y', t'_2, \ldots, t'_k)$ and $y \neq y'$. The extension to the general case is obvious. We have to distinguish the following cases.

$\underline{y \in var(\mathbf{G}) \text{ and } y' \notin var(\mathbf{G})}$ In this case we have a contradiction. In fact since ϑ is idempotent and (by hypothesis) $y \in range(\vartheta)$, $y \notin dom(\vartheta)$ and therefore, since $dom(\vartheta) = dom(\vartheta')$, $y \notin dom(\vartheta')$. Then it is easy to check that $\mathbf{G}\vartheta \not\equiv \mathbf{G}\vartheta'$.

$\underline{y \notin var(\mathbf{G}) \text{ and } y' \in var(\mathbf{G})}$ The proof is exactly the same as the proof of the previous case.

$\underline{y \in var(\mathbf{G}) \text{ and } y' \in var(\mathbf{G})}$ In this case, analogously to the first case, $y, y' \notin dom(\vartheta) \cup dom(\vartheta')$. Then we prove by contradiction that $\mathbf{G}\vartheta \not\equiv \mathbf{G}\vartheta'$. By definition of variance, there exists a renaming ρ such that

$$\mathbf{G}\vartheta\rho = \mathbf{G}\vartheta'. \tag{1}$$

By our hypothesis on ϑ and ϑ', $\{y/y'\} \subseteq \rho$ and then, since $y' \notin dom(\vartheta)$ and $y \in var(\mathbf{G})$, $\{y/y'\} \subseteq (\vartheta\rho)|_{\mathbf{G}} = \vartheta'$, where the last equality follows by (1). This contradicts the fact that $y \notin dom(\vartheta')$.

$\underline{y \notin var(\mathbf{G}) \text{ and } y' \notin var(\mathbf{G})}\rfloor$ Note that the name of the variables not in $var(\mathbf{G})$ is irrelevant, since if $y \notin var(\mathbf{G})$ and $y' \notin var(\mathbf{G})$ then $(\Gamma(\vartheta))|_{var(\mathbf{G})} = (\Gamma(\vartheta\{y/y', y'/y\}))|_{var(\mathbf{G})}$. Therefore

$$(\Gamma(\vartheta))|_{var(\mathbf{G})} = (\Gamma(\vartheta'))|_{var(\mathbf{G})}.$$

$\underline{n > 0}\rfloor$ Without loss of generality, we can assume that there exists $x \in dom(\vartheta) \setminus dom(\vartheta')$. Since (by hypothesis) $dom(\vartheta) \subseteq var(\mathbf{G})$, $x \in var(\mathbf{G})$ and, since $\mathbf{G}\vartheta \equiv \mathbf{G}\vartheta'$, there exists a variable y such that $\{x/y\} \subseteq \vartheta$. Then we have two possibilities.

$\underline{y \notin var(\mathbf{G})}\rfloor$ In this case, by definition of Γ, $(\Gamma(\vartheta))|_{var(\mathbf{G})} = ((x \leftrightarrow y) \wedge \Gamma(\vartheta|_{dom(\vartheta)\setminus\{x\}}))|_{var(\mathbf{G})}$ and then, by definition of logical equivalence,

$$\begin{aligned}(\Gamma(\vartheta))|_{var(\mathbf{G})} = ((x \leftrightarrow y) \wedge \\ ((\Gamma(\vartheta|_{dom(\vartheta)\setminus\{x\}}))[y \mapsto x]))|_{var(\mathbf{G})}.\end{aligned} \tag{2}$$

Now observe that $(\Gamma(\vartheta|_{dom(\vartheta)\setminus\{x\}}))[y \mapsto x]$ is the formula $\Gamma(\vartheta'')$, associated to the idempotent substitution ϑ'', obtained by replacing the occurrences of the variable y with x in the substitution $\vartheta|_{dom(\vartheta)\setminus\{x\}}$. Moreover, by (2),

$$\begin{aligned}(\Gamma(\vartheta))|_{var(\mathbf{G})} = (((x \leftrightarrow \mathit{true}) \wedge \Gamma(\vartheta'')) \vee ((x \leftrightarrow \mathit{false}) \\ \wedge \Gamma(\vartheta'')))|_{var(\mathbf{G})} = (\Gamma(\vartheta''))|_{var(\mathbf{G})}.\end{aligned} \tag{3}$$

By construction $\mathbf{G}\vartheta \equiv \mathbf{G}\vartheta''$ and therefore $\mathbf{G}\vartheta'' \equiv \mathbf{G}\vartheta'$. Moreover, since $x \notin dom(\vartheta') \cup dom(\vartheta'')$ and $dom(\vartheta'') = dom(\vartheta) \setminus \{x\}$, $card(dom(\vartheta') \setminus dom(\vartheta'')) + card(dom(\vartheta'') \setminus dom(\vartheta')) = n-1$. Then, by inductive hypothesis, $(\Gamma(\vartheta'))|_{var(\mathbf{G})} = (\Gamma(\vartheta''))|_{var(\mathbf{G})}$ and therefore, by (3), $(\Gamma(\vartheta))|_{var(\mathbf{G})} = (\Gamma(\vartheta'))|_{var(\mathbf{G})}$.

$\underline{y \in var(\mathbf{G})}\rfloor$ Since $\mathbf{G}\vartheta \equiv \mathbf{G}\vartheta'$, there exists a renaming ρ such that $\mathbf{G}\vartheta\rho = \mathbf{G}\vartheta'$ and, since ϑ' is idempotent, $(\vartheta\rho)|_{\mathbf{G}}$ is also idempotent. Now observe that, since $x \in dom(\vartheta) \setminus dom(\vartheta')$, $(\vartheta\rho)|_{\mathbf{G}} = \vartheta'$ and by definition of composition, $\{y/x\} \subseteq (\vartheta\rho)|_{\mathbf{G}} = \vartheta'$.

Let $\vartheta'' := \{y/x\}\bar{\vartheta}$, where $\bar{\vartheta}$ is obtained by replacing any occurrence of y with x in the substitution $\vartheta|_{dom(\vartheta)\setminus\{x\}}$. By definition of Γ,

$$\Gamma(\vartheta) = (x \leftrightarrow y) \wedge \Gamma(\vartheta|_{dom(\vartheta)\setminus\{x\}})$$

$$= (x \leftrightarrow y) \wedge ((\Gamma(\vartheta|_{dom(\vartheta)\setminus\{x\}}))[y \mapsto x])$$
$$= (x \leftrightarrow y) \wedge \Gamma(\bar{\vartheta}).$$

Hence

$$\Gamma(\vartheta) = \Gamma(\vartheta''). \tag{4}$$

By construction $\mathbf{G}\vartheta \equiv \mathbf{G}\vartheta''$ and then $\mathbf{G}\vartheta'' \equiv \mathbf{G}\vartheta'$. Moreover, by definition of ϑ'', $dom(\vartheta'') = (dom(\vartheta) \cup \{y\}) \setminus \{x\}$ and then $card(dom(\vartheta') \setminus dom(\vartheta'')) + card(dom(\vartheta'') \setminus dom(\vartheta')) = n-2$. Then, by inductive hypothesis, $(\Gamma(\vartheta'))|_{var(\mathbf{G})} = (\Gamma(\vartheta''))|_{var(\mathbf{G})}$ and therefore, by (4), $(\Gamma(\vartheta))|_{var(\mathbf{G})} = (\Gamma(\vartheta'))|_{var(\mathbf{G})}$. ■

Proof of Section 3.5.1. We have to prove several facts.

ψ is an observable The proof that there exists τ_Γ^γ such that $\langle \tau_\Gamma, \tau_\Gamma^\gamma \rangle : \mathbb{A}_{ca} \rightleftharpoons \mathbb{A}_{gr}$ is a Galois insertion, is straightforward by definition of τ_Γ. Then, we can define $\psi^\gamma : \mathbb{A}_{gr} \to \mathbb{C}$ as $\psi^\gamma = \xi^\gamma \circ \tau_\Gamma^\gamma$. It is easy to check that ψ maps finite elements to finite elements and that $\langle \psi, \psi^\gamma \rangle$ satisfies Points 1 and 2 of Definition 3.1.1.

Now, let $D, D' \in \mathbb{PC}$ and $D \equiv_{\mathbb{C}} D'$. By definition, $\psi = \tau_\Gamma \xi$ and, by the proof that ξ satisfies Point 1 of Definition 3.1.1 (see proof of Example 3.1.3), $\xi^\gamma \xi(D) \equiv_{\mathbb{C}} \xi^\gamma \xi(D')$, which (by definition of $\equiv_{\mathbb{A}_{ca}}$) is equivalent to $\xi(D) \equiv_{\mathbb{A}_{ca}} \xi(D')$. Then it suffices (by taking $X = \xi(D)$ and $X' = \xi(D')$) to prove that, for any pair of pure $\mathbb{A}$-collections $X, X' \in \mathbb{A}_{ca}$, $X \equiv_{\mathbb{A}_{ca}} X' \Longrightarrow \tau_\Gamma^\gamma \tau_\Gamma(X) \equiv_{\mathbb{A}_{ca}} \tau_\Gamma^\gamma \tau_\Gamma(X')$.

First of all observe that the following equivalences hold.

1. By definition of τ_Γ, for any goal $\mathbf{G}$ and for any idempotent substitution ϑ such that $dom(\vartheta) \subseteq var(\mathbf{G})$,

$$\vartheta \in \tau_\Gamma^\gamma \tau_\Gamma(X)(\mathbf{G}) \Longleftrightarrow (\Gamma(\vartheta))|_{var(\mathbf{G})} \to \tau_\Gamma(X)(\mathbf{G}), \tag{1}$$

   where $\to$ denotes the logical implication.

2. By Corollary 1.3.5, for any substitution ϑ such that $dom(\vartheta) \subseteq \mathbf{x}$ and for any renaming ρ, $\rho^{-1}\vartheta\rho = \{x\rho/t\rho \mid x/t \in \vartheta\}$. Then $\Gamma(\rho^{-1}\vartheta\rho) = \Gamma(\vartheta)\rho$ and therefore

$$(\Gamma(\rho^{-1}\vartheta\rho))|_{\mathbf{x}\rho} = ((\Gamma(\vartheta))|_{\mathbf{x}})\rho. \tag{2}$$

3. By using the same arguments of the proof of Example 3.1.3, for any pair of pure $\mathbb{A}$-collections $X, X' \in \mathbb{A}_{ca}$, $X \equiv_{\mathbb{A}_{ca}} X'$ if and only if for any $p(\mathbf{x}) \in$

Goals, there exists a renaming ρ such that, if $\mathsf{X}(\mathsf{p}(\mathbf{x}))$ is defined, then $\mathsf{X}'(\mathsf{p}(\mathbf{x})\rho)$ is defined and, for any $\bar{\vartheta} \in \mathsf{X}(\mathsf{p}(\mathbf{x}))$, there exists $\bar{\vartheta}' \in \mathsf{X}'(\mathsf{p}(\mathbf{x})\rho)$ such that $\mathsf{p}(\mathbf{x})\bar{\vartheta} \equiv \mathsf{p}(\mathbf{x})\rho\bar{\vartheta}'$ and vice versa. Then, by definition of renaming, there exists a renaming ρ', such that $\mathsf{p}(\mathbf{x})\rho\rho^{-1}\bar{\vartheta}\rho\rho' = \mathsf{p}(\mathbf{x})\rho\bar{\vartheta}'$ and therefore, by Lemma 3.9.3 and by (2), $(\Gamma(\bar{\vartheta}'))|_{\mathbf{x}\rho} = (\Gamma(\rho^{-1}\bar{\vartheta}\rho\rho'))|_{\mathbf{x}\rho} = (\Gamma(\rho^{-1}\bar{\vartheta}\rho))|_{\mathbf{x}\rho} = ((\Gamma(\bar{\vartheta}))|_{\mathbf{x}})\rho$. Then, by definition of τ_Γ, $\mathsf{X} \equiv_{\mathbb{A}_{ca}} \mathsf{X}'$ implies that, for any $\mathsf{p}(\mathbf{x}) \in$ *Goals*, there exists a renaming ρ such that, if $\mathsf{X}(\mathsf{p}(\mathbf{x}))$ is defined, then $\mathsf{X}'(\mathsf{p}(\mathbf{x})\rho)$ is defined and $\tau_\Gamma(\mathsf{X}')(\mathsf{p}(\mathbf{x})\rho) = (\tau_\Gamma(\mathsf{X})(\mathsf{p}(\mathbf{x})))\rho$.

Now let $\mathsf{X}, \mathsf{X}' \in \mathbb{A}_{ca}$ two pure $\mathbb{A}$-collections such that $\mathsf{X} \equiv_{\mathbb{A}_{ca}} \mathsf{X}'$. By definition of $\equiv_{\mathbb{A}_{ca}}$ and by Point 3, we have only to prove that, for any goal $\mathsf{p}(\mathbf{x})$ such that $\tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X})(\mathsf{p}(\mathbf{x}))$ is defined, there exists a renaming ρ such that $\tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X}')(\mathsf{p}(\mathbf{x})\rho)$ is defined and, for any $\vartheta \in \tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X})(\mathsf{p}(\mathbf{x}))$, there exists $\vartheta' \in \tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X}')(\mathsf{p}(\mathbf{x})\rho)$ such that $\mathsf{p}(\mathbf{x})\vartheta \equiv \mathsf{p}(\mathbf{x})\rho\vartheta'$. Then the thesis follows by symmetry.

Let $\mathsf{p}(\mathbf{x}) \in$ *Goals* such that $\tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X})(\mathsf{p}(\mathbf{x}))$ is defined. By definition of τ_Γ, $\mathsf{X}(\mathsf{p}(\mathbf{x}))$ is also defined. Then, by Point 3 and since (by hypothesis) $\mathsf{X} \equiv_{\mathbb{A}_{ca}} \mathsf{X}'$, there exists a renaming ρ such that $\mathsf{X}'(\mathsf{p}(\mathbf{x})\rho)$ is defined and

$$\tau_\Gamma(\mathsf{X}')(\mathsf{p}(\mathbf{x})\rho) = (\tau_\Gamma(\mathsf{X})(\mathsf{p}(\mathbf{x})))\rho. \tag{3}$$

By definition of τ_Γ and since $\mathsf{X}'(\mathsf{p}(\mathbf{x})\rho)$ is defined, $\tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X}')(\mathsf{p}(\mathbf{x})\rho)$ is also defined. Let $\vartheta \in \tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X})(\mathsf{p}(\mathbf{x}))$. By Corollary 1.3.5, $\rho^{-1}\vartheta\rho$ is idempotent and, since $\vartheta \in \tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X})(\mathsf{p}(\mathbf{x}))$, $dom(\rho^{-1}\vartheta\rho) = dom(\vartheta)\rho \subseteq var(\mathsf{p}(\mathbf{x})\rho)$. Then to prove the thesis it is sufficient to prove that $\rho^{-1}\vartheta\rho \in \tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X}')(\mathsf{p}(\mathbf{x})\rho)$.

By Point 1 and since $\vartheta \in \tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X})(\mathsf{p}(\mathbf{x}))$, $(\Gamma(\vartheta))|_{\mathbf{x}} \to \tau_\Gamma(\mathsf{X})(\mathsf{p}(\mathbf{x}))$ and therefore, by Point 2 and (3), $(\Gamma(\rho^{-1}\vartheta\rho))|_{\mathbf{x}\rho} = ((\Gamma(\vartheta))|_{\mathbf{x}})\rho \to (\tau_\Gamma(\mathsf{X})(\mathsf{p}(\mathbf{x})))\rho = \tau_\Gamma(\mathsf{X}')(\mathsf{p}(\mathbf{x})\rho)$. Then, by Point 1, $\rho^{-1}\vartheta\rho \in \tau_\Gamma^\gamma\tau_\Gamma(\mathsf{X}')(\mathsf{p}(\mathbf{x})\rho)$ and then the thesis.

abstract operations Let $\cdot_\xi$ be the abstract instantiation operation corresponding to the ξ observable. The abstract instantiation w.r.t. ψ can be defined in terms of τ_Γ and $\cdot_\xi$ since $\mathsf{A} \tilde{\cdot} \mathsf{X} = \psi(\mathsf{A} \cdot \psi^\gamma(\mathsf{X})) = \tau_\Gamma\xi(\mathsf{A} \cdot \xi^\gamma\tau_\Gamma^\gamma(\mathsf{X})) = \tau_\Gamma(\mathsf{A} \cdot_\xi \tau_\Gamma^\gamma(\mathsf{X}))$. Hence the definition of the abstract operation can be computed in terms of the corresponding operations on $\mathbb{A}_{ca}$, which have been already computed in Section 3.4.1. The same holds for $\widetilde{\sum}$, $\tilde{\times}$ and $\tilde{\triangleright}$.

We prove only the correctness of $\tilde{\cdot}$. The proof of the correctness of $\tilde{\times}$ is analogous, while the proof of the correctness of $\widetilde{\sum}$ is straightforward.

First of all note that the following facts hold.

1. By definition of τ_Γ, a substitution $\sigma \in \tau_\Gamma^\gamma(X)(A')$ if and only if $dom(\sigma) \subseteq var(A')$ and $(\Gamma(\sigma))|_{var(A')} \to X(A')$. Then, since (by definition of τ_Γ) $var(X(A')) \subseteq var(A')$, $\langle H, \Theta' \rangle$ is a renamed version of $\langle A', \tau_\Gamma^\gamma(X)(A') \rangle$ if and only if $\langle H, F' \rangle$ is a renamed version of $\langle A', X(A') \rangle$ and $\vartheta' \in \Theta'$ if and only if $dom(\vartheta') \subseteq var(H)$ and $(\Gamma(\vartheta'))|_{var(H)} \to F'$. Moreover, if $F_1 \to F'$ and $var(F_1) \subseteq var(H)$, then there exists a substitution ϑ' such that $(\Gamma(\vartheta'))|_{var(H)} = F_1$ and therefore, by definition of τ_Γ^γ, $\vartheta' \in \Theta'$.
2. Let H be an atom and δ, ϑ' be idempotent substitutions such that there exists $\vartheta = (mgu(H\delta, H\vartheta'))|_{H\delta}$ and $var(H\delta) \cap var(H\vartheta') = \emptyset$, $dom(\delta) \subseteq var(H)$. Then, by definition of Γ,
$$(\Gamma(\vartheta))|_{var(H\delta)} = ((\Gamma(\vartheta'))|_{var(H)} \wedge \Gamma(\delta))|_{var(H\delta)}.$$
3. Let $\mathbf{x}$ be variables and $\{F_i\}_{i \in I}$ be a set of formulas such that $\bigcup_{i \in I}(var(F_i))$ is a finite set of variables. Then, by a straightforward inductive argument on $card(\bigcup_{i \in I}(var(F_i)) \setminus \{\mathbf{x}\})$, $\bigvee_{i \in I}(F_i|_{\mathbf{x}}) = (\bigvee_{i \in I} F_i)|_{\mathbf{x}}$.

Now we can prove that the $\tilde{\cdot}$ operation is correctly defined.

$A \tilde{\cdot} X$

$= $ [by the first observation and by definition of τ_Γ]

$\phi[F/A]$ where $F = \bigvee\{(\Gamma(\vartheta))|_{var(A)} \mid \langle H, \Theta' \rangle$ is renamed apart (from A) version of $\langle A', \tau_\Gamma^\gamma(X)(A') \rangle$, for some $A' \leq A$, $\vartheta' \in \Theta'$, there exists δ s.t. $A = H\delta$ and $\vartheta = (mgu(A, H\vartheta'))|_{var(A)}\}$

$= $ [by Points 1 and 2 and since $A = H\delta$]

$\phi[F/A]$ where $F = \bigvee\{F'' \mid \langle H, F' \rangle$ is renamed apart (from A) version of $\langle A', X(A') \rangle$, for some $A' \leq A$, $F_i \to F'$, $var(F_i) \subseteq var(H)$, there exists δ s.t. $A = H\delta$, $dom(\delta) \subseteq var(H)$ and $F'' = (F_i \wedge \Gamma(\delta))|_{var(A)}\}$

$= $ [by logic properties]

$\phi[F/A]$ where $F = \bigvee\{F'' \mid \langle H, F' \rangle$ is renamed apart (from A) version of $\langle A', X(A') \rangle$, for some $A' \leq A$, there exists δ s.t. $A = H\delta$, $dom(\delta) \subseteq var(H)$ and $F'' = \bigvee_{F_i \to F', var(F_i) \subseteq var(H)} (F_i \wedge \Gamma(\delta))|_{var(A)}\}$

$= $ [by Point 3 and since $\wedge$ is distributive on $\vee$]

$\phi[F/A]$ where $F = \bigvee\{F'' \mid \langle H, F' \rangle$ is renamed apart (from A) version of $\langle A', X(A') \rangle$, for some $A' \leq A$, there exists δ s.t. $A = H\delta$, $dom(\delta) \subseteq var(H)$ and $F'' = ((\bigvee_{F_i \to F', var(F_i) \subseteq var(H)} F_i) \wedge \Gamma(\delta))|_{var(A)}\}$

$= $ [since $var(F') \subseteq var(H)$ and by logical properties]

$\phi\left[{}^{F}/_{A}\right]$ where $F = \bigvee\{F'' \mid \langle H, F'\rangle$ is renamed apart (from A) version of $\langle A', X(A')\rangle$, for some $A' \leq A$, there exists δ s.t. $A = H\delta$, $dom(\delta) \subseteq var(H)$ and $F'' = (F' \wedge \Gamma(\delta))|_{var(A)}\}$

ψ is a semi-denotational observable Let $A \in Atoms$, $D', D'' \in \mathbb{C}$, $D \in \mathbb{PC}$, $\mathbf{G} \in Goals$ and let $\{D_j\}_{j\in J} \subseteq \mathbb{PC}$ be a chain. First of all note that, for any goal $\mathbf{G}$,

$$\psi(Id_{\mathbb{I}}) = \lambda p(\mathbf{x}).\, false \text{ and } \psi(\phi_{\mathbf{G}}) = \phi\left[{}^{false}/_{\mathbf{G}}\right]. \tag{4}$$

Then the following equivalences hold.

Axiom (3.5.1) The proof is straightforward by observing that $\wedge$ is *left/right distributive* on $\vee$.

Axiom (3.5.2) By (4) and by definition of abstract operators,

$$\psi(A \cdot \psi^{\gamma}\psi(Id_{\mathbb{I}})) = A \tilde{\cdot} \psi(Id_{\mathbb{I}}) = \psi(\phi_A) = \psi(A \cdot Id_{\mathbb{I}}).$$

Axioms (3.5.3), (3.5.5) and (3.5.9) We prove that

$$\psi(\psi^{\gamma}\psi(D') \triangleright \psi^{\gamma}\psi(D'')) = \psi(\psi^{\gamma}\psi(D') \triangleright D'').$$

We have two cases.

$\exists \mathbf{B} \in \boldsymbol{Goals}, \mathtt{d}'' \in \mathbf{D}''(\mathbf{B}).\, \mathbf{B} \neq \square$ and $\boldsymbol{last}(\mathtt{d}'') = \square$ First of all observe that for any goal $\mathbf{G}$, $\psi(\psi^{\gamma}\psi(D') \triangleright \psi^{\gamma}\psi(D''))(\mathbf{G})$ is defined if and only if $\psi(\psi^{\gamma}\psi(D') \triangleright D'')(\mathbf{G})$ is defined if and only if $D'(\mathbf{G})$ is defined.

Now, let us consider a goal $\mathbf{G} = p_1(\mathbf{t}_1), \ldots, p_n(\mathbf{t}_n)$ such that $D'(\mathbf{G})$ is defined. By definition of ψ, the derivation

$$\mathtt{d} = \mathbf{G} \xrightarrow[p_1(\mathbf{x}_1)]{\{\mathbf{x}_1/\mathbf{t}_1\}} \cdots \xrightarrow[p_n(\mathbf{x}_n) \leftarrow \mathbf{B}']{\{\mathbf{x}_n/\mathbf{t}_n\}} \mathbf{B}' \in \psi^{\gamma}\psi(D'),$$

where $\mathbf{B}'$ is a renamed version of $\mathbf{B}$, such that $var(\mathbf{B}') \cap var(\mathbf{G}) = \emptyset$. Then, by definition of $\triangleright$, the derivation $\mathtt{d} :: \mathtt{d}' \in (\psi^{\gamma}\psi(D') \triangleright D'')(\mathbf{G})$, where $\mathtt{d}'$ is a renamed apart (w.r.t. $\mathtt{d}$) version of $\mathtt{d}''$, and it is easy to check that $last(\mathtt{d} :: \mathtt{d}') = \square$ and $answer(\mathtt{d} :: \mathtt{d}') = \varepsilon$. Moreover, since $D'' \sqsubseteq \psi^{\gamma}\psi(D'')$, $\mathtt{d} :: \mathtt{d}' \in (\psi^{\gamma}\psi(D') \triangleright \psi^{\gamma}\psi(D''))(\mathbf{G})$.

Then, by definition of ψ, for any $\mathbf{G} \in Goals$ such that $D'(\mathbf{G})$ is defined, $\psi(\psi^{\gamma}\psi(D') \triangleright \psi^{\gamma}\psi(D''))(\mathbf{G}) = true = \psi(\psi^{\gamma}\psi(D') \triangleright D'')(\mathbf{G})$ and then the thesis.

$\forall \mathbf{B} \in \boldsymbol{Goals}.\ \nexists \mathtt{d}'' \in \mathbf{D}''(\mathbf{B}).\, \mathbf{B} \neq \square$ **and** $\boldsymbol{last}(\mathtt{d}'') = \square$ By definition of ψ, for any $\mathbf{B} \in \mathit{Goals}$ there is no $\mathtt{d}' \in \psi^{\gamma}\psi(\mathsf{D}'')(\mathbf{B})$ such that $\mathit{last}(\mathtt{d}') = \square$. Then, by definition of ψ, $\psi(\psi^{\gamma}\psi(\mathsf{D}') \triangleright \psi^{\gamma}\psi(\mathsf{D}'')) = \psi(\mathsf{D}') = \psi(\psi^{\gamma}\psi(\mathsf{D}') \triangleright \mathsf{D}'')$.

Axiom (3.5.4) By (4) and by definition of ψ, $\psi(\psi^{\gamma}\psi(\mathsf{D}') \times \psi^{\gamma}\psi(\phi_{\mathbf{G}})) = \psi(\mathsf{D}') \mathbin{\widetilde{\times}} \psi(\phi_{\mathbf{G}}) = \lambda\mathbf{G}'.\, \mathit{false}$ where $\mathbf{G}' = (\mathbf{G}_1, \mathbf{G}_2)$ and X' is a renamed version of $\psi(\mathsf{D}')$ such that $\mathsf{X}'(\mathbf{G}_1)$ is defined and $\mathbf{G}_2 \equiv \mathbf{G}$. Now, the proof is straightforward by definition of $\times$, ψ and $\widetilde{\times}$.

Axiom (3.5.6) We prove that $\psi^{\gamma}\psi(\sum\{\mathsf{D}_j\}_{j\in J}) = \sum\{\psi^{\gamma}\psi(\mathsf{D}_j)\}_{j\in J}$.

The proof of the inequality $\psi^{\gamma}\psi(\sum\{\mathsf{D}_j\}_{j\in J}) \sqsupseteq \sum\{\psi^{\gamma}\psi(\mathsf{D}_j)\}_{j\in J}$ is straightforward by (3.3.1) and since $\psi^{\gamma}\psi$ is extensive. Now, we prove the other inequality. Let $\mathbf{G}$ be a goal such that $\psi^{\gamma}\psi(\sum\{\mathsf{D}_j\}_{j\in J})(\mathbf{G})$ is defined. By definition of ψ, by definition of $\widetilde{\sum}$ and by (3.3.1)

$$\psi(\sum\{\mathsf{D}_j\}_{j\in J})(\mathbf{G}) = \bigvee\{\psi(\mathsf{D}_j)(\mathbf{G})\}_{j\in J}.$$

Now, observe that, by definition of ψ, for any $j \in J$, $\mathit{var}(\psi(\mathsf{D}_j)(\mathbf{G})) \subseteq \mathit{var}(\mathbf{G})$ and therefore $\psi(\mathsf{D}_j)(\mathbf{G})$ is (equivalent to) a finite formula. Moreover, since $\{\mathsf{D}_j\}_{j\in J}$ is a chain, we have that $\{\psi(\mathsf{D}_j)(\mathbf{G})\}_{j\in J}$ is also a chain (ordered by logical implication). Since for any $j \in J$, $\mathit{var}(\psi(\mathsf{D}_j)(\mathbf{G})) \subseteq \mathit{var}(\mathbf{G})$, the chain $\{\psi(\mathsf{D}_j)(\mathbf{G})\}_{j\in J}$ is finite and therefore, there exists $i \in J$ such that $\psi(\mathsf{D}_i)(\mathbf{G}) = \bigvee\{\psi(\mathsf{D}_j)(\mathbf{G})\}_{j\in J}$. Since the previous result holds for any $\mathbf{G} \in \mathit{Goals}$, the thesis follows by definition of $\sum$ and ψ^{γ}.

Other Axioms The proof that ψ satisfies Axioms (3.5.7) and (3.5.8) is analogous to the proof of Axioms (3.5.3), (3.5.5) and (3.5.9).

By using the same argument, we have that $\psi(\psi^{\gamma}\psi(\mathsf{D}'') \triangleright \mathit{su}(\psi^{\gamma}\psi(\mathsf{D}))) = \psi(\psi^{\gamma}\psi(\mathsf{D}'') \triangleright \psi^{\gamma}\psi(\mathit{su}(\psi^{\gamma}\psi(\mathsf{D})))) = \psi(\mathsf{D}'') \mathbin{\widetilde{\triangleright}} \psi(\mathit{su}(\psi^{\gamma}\psi(\mathsf{D})))$, where the last equality follows by definition of abstract operators. Now, by definition of ψ, $\psi(\mathsf{D}'') \mathbin{\widetilde{\triangleright}} \psi(\mathit{su}(\psi^{\gamma}\psi(\mathsf{D}))) =$

$$\begin{cases} \lambda\mathbf{G}.\, \mathit{true}|_{\mathsf{D}''(\mathbf{G})\ \mathrm{def}} & \text{if } \exists \mathbf{G}', \mathtt{d}' \in \mathit{su}(\psi^{\gamma}\psi(\mathsf{D}))(\mathbf{G}').\ \mathit{last}(\mathtt{d}') = \square \\ \psi(\mathsf{D}'') & \text{otherwise} \end{cases}$$

where $\lambda\mathbf{G}.\, \mathit{true}|_{\mathsf{X}(\mathbf{G})\ \mathrm{def}}$ denotes the function which, on input $\mathbf{G}$, assumes the value *true* if $\mathsf{X}(\mathbf{G})$ is defined and is otherwise undefined. Moreover $\psi^{\gamma}\psi(\mathsf{D}'') \triangleright \mathit{su}(\psi^{\gamma}\psi(\mathsf{D})) =$

$$\begin{cases} \lambda\mathbf{G}.\, \mathsf{S}_{\mathbf{G}}|_{\mathsf{D}''(\mathbf{G})\ \mathrm{def}} & \text{if } \exists \mathbf{G}', \mathtt{d}' \in \mathit{su}(\psi^{\gamma}\psi(\mathsf{D}))(\mathbf{G}').\ \mathit{last}(\mathtt{d}') = \square \\ \psi^{\gamma}\psi(\mathsf{D}'') & \text{otherwise} \end{cases}$$

where, for any $\mathbf{G}$, there exists $\mathsf{d} \in \mathsf{S}_{\mathbf{G}}$, such that $\mathit{last}(\mathsf{d}) = \square$ and $\mathit{answer}(\mathsf{d}) = \varepsilon$.

Now, the proof follows by definition of abstract operators and of ψ. ■

Proof of Section 3.5.2. We have to prove several facts.

κ is an observable The proof that there exists

$$\mu_{\mathsf{k}}^{\gamma}(\mathsf{X}) := \lambda\mathbf{G}.\{(\vartheta\sigma)|_{\mathbf{G}} \mid \vartheta \in \mathsf{X}(\mathbf{G}), \sigma \in \mathit{Subst}(\hat{\mathsf{V}} \to \mathsf{T})\}$$

such that $\langle \mu_{\mathsf{k}}, \mu_{\mathsf{k}}^{\gamma} \rangle : \mathbb{A}_{ca} \rightleftharpoons \mathbb{A}_{\lightning_{\mathsf{k}}}$ is a Galois insertion, is straightforward by definition of μ_{k}. Then, we can define $\kappa^{\gamma} : \mathbb{A}_{\lightning_{\mathsf{k}}} \to \mathbb{C}$ as $\kappa^{\gamma} = \xi^{\gamma} \circ \mu_{\mathsf{k}}^{\gamma}$. It is easy to check that κ maps finite elements to finite elements and that $\langle \kappa, \kappa^{\gamma} \rangle$ satisfies Points 1 and 2 of Definition 3.1.1.

As was the case for Section 3.5.1 we have only to prove that, for any pair of pure $\mathbb{A}$-collections $\mathsf{X}, \mathsf{X}' \in \mathbb{A}_{ca}$, $\mathsf{X} \equiv_{\mathbb{A}_{ca}} \mathsf{X}' \Longrightarrow \mu_{\mathsf{k}}^{\gamma}\mu_{\mathsf{k}}(\mathsf{X}) \equiv_{\mathbb{A}_{ca}} \mu_{\mathsf{k}}^{\gamma}\mu_{\mathsf{k}}(\mathsf{X}')$, i.e., that, for any $\vartheta \in \mu_{\mathsf{k}}^{\gamma}\mu_{\mathsf{k}}(\mathsf{X})(\mathsf{p}(\mathbf{x}))$, there exists $\vartheta' \in \mu_{\mathsf{k}}^{\gamma}\mu_{\mathsf{k}}(\mathsf{X}')(\mathsf{p}(\mathbf{y}))$ such that $\mathsf{p}(\mathbf{x})\vartheta \equiv \mathsf{p}(\mathbf{y})\vartheta'$. By $\mu_{\mathsf{k}}^{\gamma}$, μ_{k} definition there must be a substitution $\bar{\vartheta} \in \mathsf{X}(\mathsf{p}(\mathbf{x}))$ identical to ϑ except for any sub-term rooted at depth k. Moreover (by hypotheses) there must be a substitution $\bar{\vartheta}' \in \mathsf{X}'(\mathsf{p}(\mathbf{y}))$ such that $\mathsf{p}(\mathbf{x})\bar{\vartheta} \equiv \mathsf{p}(\mathbf{y})\bar{\vartheta}'$. Now we can easily build a substitution $\vartheta' \in \mu_{\mathsf{k}}^{\gamma}\mu_{\mathsf{k}}(\mathsf{X}')(\mathsf{p}(\mathbf{y}))$, identical to $\bar{\vartheta}'$ except for any sub-term rooted at depth k, such that $\mathsf{p}(\mathbf{x})\vartheta \equiv \mathsf{p}(\mathbf{y})\vartheta'$.

abstract operations As was the case for Section 3.5.1 the definition of the abstract operations can be computed in terms of the corresponding operations on $\mathbb{A}_{ca}$, which have been already computed in Section 3.4.1.

Consider the $\tilde{\cdot}$ operation and let $\mathsf{X}' := \phi\left[{}^{\Theta}/_{\mathsf{A}}\right]$, where $\Theta := \{\vartheta\lightning_{\mathsf{k}} \mid \langle \mathsf{H}, \Theta' \rangle$ is a renamed apart (from A) version of $\langle \mathsf{A}', \mathsf{X}(\mathsf{A}') \rangle$, for some $\mathsf{A}' \leq \mathsf{A}$, $\vartheta' \in \Theta'$, $\vartheta = \mathit{mgu}(\mathsf{A}, \mathsf{H}\vartheta')|_{\mathsf{A}}\}$. It suffices to prove that $\mathsf{X}' = \mu_{\mathsf{k}}(\mathsf{A} \cdot_{\xi} \mu_{\mathsf{k}}^{\gamma}(\mathsf{X}))$, since $\mathsf{A}\tilde{\cdot}\mathsf{X} = \mu_{\mathsf{k}}(\mathsf{A} \cdot_{\xi} \mu_{\mathsf{k}}^{\gamma}(\mathsf{X})) = \kappa(\mathsf{A} \cdot \kappa^{\gamma}(\mathsf{X}))$.

The inclusion $\mathsf{X}' \subseteq \mu_{\mathsf{k}}(\mathsf{A} \cdot_{\xi} \mu_{\mathsf{k}}^{\gamma}(\mathsf{X}))$ is straightforward, since we can map each $\mathit{depth}(\mathsf{k})$ substitution to a concrete one by mapping each variable of $\hat{\mathsf{V}}$ with a fresh variable of V. To prove the other inclusion, let $\langle \mathsf{H}, \Theta'' \rangle$ be a renamed apart (from A) version of $\langle \mathsf{A}', \mu_{\mathsf{k}}^{\gamma}(\mathsf{X})(\mathsf{A}') \rangle$ and $\vartheta'' \in \Theta''$ such that there exists $\mathit{mgu}(\mathsf{A}, \mathsf{H}\vartheta'')$. Then there exists $\langle \mathsf{H}, \Theta''' \rangle$, renamed apart (from A) version of $\langle \mathsf{A}', \mathsf{X}(\mathsf{A}') \rangle$, $\vartheta''' \in \Theta'''$ and $\sigma : \hat{\mathsf{V}} \to \mathsf{T}$ such that $\vartheta'' = (\vartheta'''\sigma)|_{\mathsf{A}}$ and there exists $\mathit{mgu}(\mathsf{A}, \mathsf{H}\vartheta''')$. By the properties of $\lightning_{\mathsf{k}}$, $(\mathit{mgu}(\mathsf{A}, \mathsf{H}\vartheta''')|_{\mathsf{A}})\lightning_{\mathsf{k}} = (\mathit{mgu}(\mathsf{A}, \mathsf{H}\vartheta'''\sigma)|_{\mathsf{A}})\lightning_{\mathsf{k}} = (\mathit{mgu}(\mathsf{A}, \mathsf{H}\vartheta'')|_{\mathsf{A}})\lightning_{\mathsf{k}}$. Hence $(\mathit{mgu}(\mathsf{A}, \mathsf{H}\vartheta''')|_{\mathsf{A}})\lightning_{\mathsf{k}} \in \mathsf{X}'$, because the mgu construction can build substitutions with domain in $\mathsf{V} \cup \hat{\mathsf{V}}$, but the restriction on the variables of A confines $\mathit{dom}(\vartheta)$ in V.

An analogous argumentation holds for the $\widetilde{\times}$ operation and the substitution $\vartheta = (\vartheta_1 \circ mgu(\mathbf{G}_2\vartheta_1, \mathbf{G}_2\vartheta_2))|_{\mathbf{G}}$.

κ is a semi-denotational observable Let $A \in Atoms$, $D', D'' \in \mathbb{C}$, $D \in \mathbb{PC}$, $\mathbf{G} \in Goals$ and let $\{D_j\}_{j\in J} \subseteq \mathbb{PC}$ be a chain. In the following we denote by $\Theta|_{\mathbf{G}} \subseteq Subst$ ($\Theta|_{\mathbf{G}}\lightning_k \subseteq Subst\lightning_k$) the set of all the substitution ϑ, such that $dom(\vartheta) \subseteq var(\mathbf{G})$.

First of all note that, for any goal $\mathbf{G}$,

$$\kappa(Id_{\mathbb{I}}) = \lambda p(\mathbf{x}).\emptyset \text{ and } \kappa(\phi_{\mathbf{G}}) = \phi\left[{}^{\emptyset}/_{\mathbf{G}}\right]. \tag{1}$$

Then the following equivalences hold.

Axiom (3.5.1) Immediate.

Axiom (3.5.2) By (1) and by definition of abstract operators,

$$\kappa(A \cdot \kappa^{\gamma}\kappa(Id_{\mathbb{I}})) = A \tilde{\cdot} \kappa(Id_{\mathbb{I}}) = \kappa(\phi_A) = \kappa(A \cdot Id_{\mathbb{I}}).$$

Axioms (3.5.3), (3.5.5) and (3.5.9) We prove that $\kappa(\kappa^{\gamma}\kappa(D') \triangleright \kappa^{\gamma}\kappa(D'')) = \kappa(\kappa^{\gamma}\kappa(D') \triangleright D'')$. We have two cases.

$\exists \mathbf{B} \in \boldsymbol{Goals}, \mathtt{d}'' \in \mathbf{D}''(\mathbf{B}).\, \mathbf{B} \neq \square$ and $\boldsymbol{last}(\mathtt{d}'') = \square$ First of all observe that for any goal $\mathbf{G}$, $\kappa(\kappa^{\gamma}\kappa(D') \triangleright \kappa^{\gamma}\kappa(D''))(\mathbf{G})$ is defined if and only if $\kappa(\kappa^{\gamma}\kappa(D') \triangleright D'')(\mathbf{G})$ is defined if and only if $D'(\mathbf{G})$ is defined.

Now, let us consider a goal $\mathbf{G}$ such that $D'(\mathbf{G})$ is defined. By definition of κ, for any substitution $\vartheta \in \Theta|_{\mathbf{G}}$, there exists a derivation $\mathtt{d} \in \kappa^{\gamma}\kappa(D')$, such that $last(\mathtt{d}) = \mathbf{B}'$, $answer(\mathtt{d}) = \vartheta$ and $\mathbf{B}'$ is a renamed version of $\mathbf{B}$, such that $var(\mathbf{B}') \cap var(\mathbf{G}) = \emptyset$. Then, by definition of $\triangleright$, for any substitution $\vartheta \in \Theta|_{\mathbf{G}}$, there exists a derivation $\mathtt{d} :: \mathtt{d}' \in (\kappa^{\gamma}\kappa(D') \triangleright D'')(\mathbf{G})$, where $\mathtt{d}'$ is a renamed apart (w.r.t. $\mathtt{d}$) version of $\mathtt{d}''$, such that $answer(\mathtt{d} :: \mathtt{d}') = \vartheta$ and $last(\mathtt{d} :: \mathtt{d}') = \square$. Moreover, since $D'' \sqsubseteq \kappa^{\gamma}\kappa(D'')$, $\mathtt{d} :: \mathtt{d}' \in (\kappa^{\gamma}\kappa(D') \triangleright \kappa^{\gamma}\kappa(D''))(\mathbf{G})$.

Then, by definition of κ, we have that for any $\mathbf{G} \in Goals$ such that $D'(\mathbf{G})$ is defined, $\kappa(\kappa^{\gamma}\kappa(D') \triangleright \kappa^{\gamma}\kappa(D''))(\mathbf{G}) = \Theta|_{\mathbf{G}}\lightning_k = \kappa(\kappa^{\gamma}\kappa(D') \triangleright D'')$ and then the thesis.

$\forall \mathbf{B} \in \boldsymbol{Goals}.\ \nexists \mathtt{d}'' \in \mathbf{D}''(\mathbf{B}).\, \mathbf{B} \neq \square$ and $\boldsymbol{last}(\mathtt{d}'') = \square$ By definition of κ, for any $\mathbf{B} \in Goals$ there is no $\mathtt{d}' \in \kappa^{\gamma}\kappa(D'')(\mathbf{B})$ such that $last(\mathtt{d}') = \square$. Then, by definition of κ, $\kappa(\kappa^{\gamma}\kappa(D') \triangleright \kappa^{\gamma}\kappa(D'')) = \kappa(D') = \kappa(\kappa^{\gamma}\kappa(D') \triangleright D'')$.

Axiom (3.5.4) By (1) and by definition of κ,

$$\kappa(\kappa^{\gamma}\kappa(D') \times \kappa^{\gamma}\kappa(\phi_{\mathbf{G}})) = \kappa(D') \widetilde{\times} \kappa(\phi_{\mathbf{G}}) = \lambda \mathbf{G}'.\emptyset$$

where $\mathbf{G}' = (\mathbf{G}_1, \mathbf{G}_2)$ and X' is a renamed version of $\kappa(D')$ such that $X'(\mathbf{G}_1)$ is defined and $\mathbf{G}_2 \equiv \mathbf{G}$. Now, the proof is straightforward by definition of $\times$, κ and $\widetilde{\times}$.

Axiom (3.5.6) We prove that $\kappa^{\gamma}\kappa(\sum\{D_j\}_{j\in J}) = \sum\{\kappa^{\gamma}\kappa(D_j)\}_{j\in J}$.

The proof of the inequality $\kappa^{\gamma}\kappa(\sum\{D_j\}_{j\in J}) \sqsupseteq \sum\{\kappa^{\gamma}\kappa(D_j)\}_{j\in J}$ is straightforward by (3.3.1) and since $\kappa^{\gamma}\kappa$ is extensive. Now, we prove the other inequality. Let $\mathbf{G}$ be a goal such that $\kappa^{\gamma}\kappa(\sum\{D_j\}_{j\in J})(\mathbf{G})$ is defined. By definition of κ and of $\widetilde{\sum}$ and by (3.3.1)

$$\kappa(\sum\{D_j\}_{j\in J})(\mathbf{G}) = \bigcup\{\kappa(D_j)(\mathbf{G})\}_{j\in J}.$$

Now, observe that, by definition of $\bigcup$, for any $\vartheta \in \kappa(\sum\{D_j\}_{j\in J})(\mathbf{G})$ there exists $i \in J$ such that $\vartheta \in \kappa(D_i)(\mathbf{G})$. Since the previous result holds for any $\mathbf{G} \in \mathit{Goals}$, the thesis follows by definition of $\sum$ and κ^{γ}.

Other Axioms The proof that κ satisfies Axioms (3.5.7) and (3.5.8) is analogous to the proof of Axioms (3.5.3), (3.5.5) and (3.5.9). By using the same argument, we have that $\kappa(\kappa^{\gamma}\kappa(D'') \triangleright su(\kappa^{\gamma}\kappa(D))) = \kappa(\kappa(D'') \triangleright \kappa^{\gamma}\kappa(su(\kappa^{\gamma}\kappa(D)))) = \kappa(D'') \widetilde{\triangleright} \kappa(su(\kappa^{\gamma}\kappa(D)))$, where the last equality follows by definition of abstract operators. Now, by definition of κ, $\kappa(D'') \widetilde{\triangleright} \kappa(su(\kappa^{\gamma}\kappa(D))) =$

$$\begin{cases} \lambda \mathbf{G}.\, \Theta|_{\mathbf{G}} \lightning_{\kappa} |_{D''(\mathbf{G})\ \mathrm{def}} & \text{if } \exists \mathbf{G}', d' \in su(\kappa^{\gamma}\kappa(D))(\mathbf{G}').\ \mathit{last}(d') = \square \\ \kappa(D'') & \text{otherwise} \end{cases}$$

where $\lambda \mathbf{G}.\, S|_{X(\mathbf{G})\ \mathrm{def}}$ denotes the function which, on input $\mathbf{G}$, assumes the value S if $X(\mathbf{G})$ is defined and is otherwise undefined. Moreover $\kappa^{\gamma}\kappa(D'') \triangleright su(\kappa^{\gamma}\kappa(D)) =$

$$\begin{cases} \lambda \mathbf{G}.\, \Theta|_{\mathbf{G}} |_{D''(\mathbf{G})\ \mathrm{def}} & \text{if } \exists \mathbf{G}', d' \in su(\kappa^{\gamma}\kappa(D))(\mathbf{G}').\ \mathit{last}(d') = \square \\ \kappa^{\gamma}\kappa(D'') & \text{otherwise.} \end{cases}$$

Now, the proof follows by definition of abstract operators and of κ. ■

Part II

Abstract Diagnosis

Chapter 4

Semantic Framework for Applications

As already pointed out in Section 3.7, in practical applications it is often the case that a denotational definition is compared to an operational definition modeling a different observable. For example, when one is concerned with computed answers (or their ground instances),

- the operational semantics is usually defined by *SLD*-resolution, i.e., by a transition system for the observable *SLD*-trees. Computed answers (or their ground instances) are then obtained by abstracting the final result;

- the denotational semantics is defined directly on the abstract domain by taking the least fixpoint of the s-semantics immediate consequence operator (or of the standard ground T_P operator).

As a consequence, the operational semantics is "more expressive" and yet contains more information. For example, from this operational semantics designed for computed answers, we can obtain information about call patterns, since both computed answers and call patterns are abstractions of the observable we are indeed modeling.

The relation between operational and denotational definitions that we have just noticed for precise observables, holds for non-precise observables too. Consider now the case of groundness analysis, based on the abstract domain $\mathcal{POS}$. As in the case of precise observables, the operational and denotational definitions use different observables. Namely,

- the (top-down) abstract operational semantics models a semi-perfect observable (the one of Section 3.6.1). Observations (i.e., groundness dependencies for computed answers) are then obtained by abstracting the result;

- the (bottom-up) abstract denotational semantics models directly a semi-denotational observable (the one of Section 3.5.1).

As in the case of precise observables, the two definitions are equivalent from the viewpoint of precision. However, since the top-down abstract semantics is more concrete, it allows to derive more information (e.g., groundness dependencies of call patterns).

The problem can be formulated in a formal setting. In both cases

- the abstract denotational semantics essentially models an abstraction τ of the computed answers observable ξ.
- the abstract operational semantics is obtained by computing *SLD*-trees on a domain of abstract terms and then collecting all the (abstract) computed answers.

Hence, the computation on the abstract domain are essentially computations w.r.t. a semi-perfect observable τ' which, when applied to a refutation $\mathtt{d}$, gives the same results as τ applied to $\mathit{answer}(\mathtt{d})$. The final collecting phase of the abstract computed answers is then the application of (the abstract version ξ' of) ξ to the abstract derivations. The reason why we need to do this, instead of using the abstract operational semantics $\mathcal{O}_{\tau\circ\xi}[\![\cdot]\!]$ of the observable $\tau\circ\xi$, is that this observable is semi-denotational and then its operational semantics is too imprecise. However, both denotations are obtained by applying a standard construction to the term abstraction τ. Indeed, for any program P, the bottom-up semantics is $\mathcal{F}_{\tau\circ\xi}[\![\mathsf{P}]\!]$ while the top-down semantics is $\xi'(\mathcal{O}_{\tau'}[\![\mathsf{P}]\!])$.

It would be useful, from the practical point of view, to systematically build such denotations starting from the term abstraction τ (which, by abuse of language, will be called observable) without the need to define the two different observables. This is even more important for the theory of abstract diagnosis since in the diagnosis case we are only concerned with observables which are abstractions of the computed answer observable. The original formulation of abstract diagnosis [25, 27], which used the axioms of the framework of Part I to classify the observables, can be reformulated in terms of simpler axioms on τ. We would choose to specialize the constructions of the two observables within this formal setting to obtain the new axioms on τ. However, in order for this part of the thesis to be self-contained, we prefer to introduce the new framework from scratch, using the ideas coming from the original one.

The new ingredients are a (different) concrete semantics and an abstraction τ of computed answers. The concrete denotational semantics models computed answers and the concrete operational semantics is obtained by computing *SLD*-trees and

then collecting all the computed answers. The two semantics are condensing and equivalent. For technical reasons that will become evident in the following, we represent computed answers by normalized equation sets.

The abstract transition system and the abstract denotational definition are systematically derived from the collecting semantics. In particular,

- the abstract denotational semantics models the abstract computed answers (according to the observable τ),
- the abstract operational semantics is obtained by computing *SLD*-trees on a domain of abstract terms (by using the abstract transition system) and then collecting all the abstract computed answers.

In the resulting taxonomy only two classes of observables can arise, depending on the precision degree.

The first class we consider is the one of *complete observables* (Section 4.3). This class corresponds to precise term abstractions (the two corresponding observables are denotational and perfect). We prove that complete observables satisfy all the properties of the concrete semantics. In particular, they are condensing and the abstract top-down and bottom-up semantics are equivalent. The class of complete observables includes the observables *correct answers* and *ground instances of computed answers*, whose semantics are the atomic logical consequence semantics (c-semantics) and the least Herbrand model, respectively.

The second class is the one of *approximate observables* (Section 4.4). This class correspond to non-precise term abstractions (the two corresponding observables are semi-denotational and semi-perfect). We prove that approximate observables are condensing and that the abstract top-down and bottom-up semantics are equivalent. The class of approximate observables includes the observables *groundness dependencies of computed answers* and *depth*(k)*-answers*. Let us just note that approximate observables are *exactly* the observables which model top-down abstract interpretation frameworks (for example, [15, 69]).

Condensing is a compositionality property which tells us that the abstract semantics of a procedure call can be derived (without losing precision) from the abstract semantics of the procedure declaration. This property is needed to reconstruct goal-dependent behaviors by using a (goal-independent) property of a set of procedure declarations. It is worth noting that the observables corresponding to the declarative semantics are condensing and that the declarative semantics do indeed characterize procedure declarations. Note also that several observables used in program analysis (for mode, type and groundness analysis) are also condensing and that a non-condensing observable can systematically be transformed into a (more

concrete) condensing observable, by using domain refinement operators (see, for example, how the condensing domain $\mathcal{POS}$ for groundness analysis can be derived [89] from the non-condensing domain $\mathcal{DEF}$).

Even if up to now the reformulation of the framework seems futile, it will be clear in Chapter 5 that it is not the case. The work on abstract diagnosis in [25, 27], which was based on the full semantic framework of Part I, showed that this reformulation is really best suited for these semantic-based applications which use only abstractions of computed answers.

The results of this chapter are from [23].

4.1 The Concrete Collecting Semantics

Recall that an *equational goal* is an object of the form $\mathcal{E}, \mathbf{B}$ where $\mathbf{B}$ is a pure goal and $\mathcal{E}$ is a finite set of solved form equation sets such that $elim(\mathcal{E}) \subseteq var(\mathbf{B})$. $\{\emptyset\}, \mathbf{B}$ will be denoted simply by $\mathbf{B}$. An *equational clause* is a formula of the form $\mathsf{H} \leftarrow \mathsf{E}, \mathbf{B}$, where $(\mathsf{H}, \mathbf{B})$ is a pure goal and E is a solved form equation set such that $elim(\mathsf{E}) \subseteq var(\mathsf{H}, \mathbf{B})$. In the following, given any program clause $\mathsf{p}(\mathbf{t}) \leftarrow \mathsf{p}_1(\mathbf{t}_1), \ldots, \mathsf{p}_n(\mathbf{t}_n)$ we will *always* consider its equational form $\mathsf{p}(\mathbf{x}) \leftarrow \mathsf{E}, \mathsf{p}_1(\mathbf{x}_1), \ldots, \mathsf{p}_n(\mathbf{x}_n)$ where $\mathsf{E} = \{\mathbf{x} = \mathbf{t}, \mathbf{x}_1 = \mathbf{t}_1, \ldots, \mathbf{x}_n = \mathbf{t}_n\}$ ($\mathbf{x}, \mathbf{x}_1, \ldots, \mathbf{x}_n$ are new distinct variables).

We denote by $\mathbb{E}$ the complete lattice of sets of finite solved form equation sets, partially ordered by $\subseteq$. We define a variance equivalence on $\mathbb{E}$ as follows. For any $\mathcal{E}, \mathcal{E}' \in \mathbb{E}$ and for any tuple of variables $\mathbf{x}$, $\mathcal{E} \equiv_{\mathbf{x}} \mathcal{E}'$ if and only if, for each $\mathsf{E} \in \mathcal{E}$, there exists $\mathsf{E}' \in \mathcal{E}'$ such that $\mathbf{x}\, mgu(\mathsf{E}) \equiv \mathbf{x}\, mgu(\mathsf{E}')$ and vice versa[1]. A *collection* C is a partial function $Goals \rightharpoonup \mathbb{E}$ such that, for every $\mathbf{G} \in Goals$, if $\mathsf{C}(\mathbf{G})$ is defined, then it is a relevant set of equation sets, i.e., $\forall \mathsf{E} \in \mathsf{C}(\mathbf{G}).\ elim(\mathsf{E}) \subseteq var(\mathbf{G})$. $\mathbb{C}$ is the domain of all the collections ordered by $\sqsubseteq$, where $\mathsf{C} \sqsubseteq \mathsf{C}'$ if and only if $\forall \mathbf{G}.\, \mathsf{C}(\mathbf{G}) \subseteq \mathsf{C}'(\mathbf{G})$. The partial order on $\mathbb{C}$ formalizes the evolution of the computation process. $(\mathbb{C}, \sqsubseteq)$ is a complete lattice. $\sqcap$ and $\sqcup$ will denote the *glb* and *lub* of $\mathbb{C}$. A *pure* collection is a collection defined for pure atomic goals only.

We define the *equivalence modulo enhanced variance* $\equiv_{\mathbb{C}}$ on collections as follows. $\mathsf{C} \equiv_{\mathbb{C}} \mathsf{C}'$ if and only if, for any $\mathbf{G}$, there exists a renaming ρ such that (if $\mathsf{C}(\mathbf{G})$ is defined then $\mathsf{C}'(\mathbf{G}\rho)$ is defined and) $\mathsf{C}(\mathbf{G}) \equiv_{var(\mathbf{G})} (\mathsf{C}'(\mathbf{G}\rho))\rho^{-1}$ and vice versa. A $\mathbb{C}$-*interpretation* $\mathcal{I}$ is a pure collection modulo enhanced variance. We denote by $\mathbb{I}_{\mathbb{C}}$ the set of $\mathbb{C}$-interpretations. $(\mathbb{I}_{\mathbb{C}}, \sqsubseteq)$ is a complete lattice. We denote the equivalence class (modulo enhanced variance) of a collection σ by σ itself. Moreover, any interpretation $\mathcal{I}$ of $\mathbb{I}_{\mathbb{C}}$ is implicitly considered also as a collection obtained by

[1] Note that, for any solved form equation set E, $mgu(\mathsf{E})$ can be "trivially" computed. Moreover, the definition of $\equiv_{\mathbf{x}}$ is independent from the choice of the *mgu*, since this is unique up to renaming.

choosing an arbitrary representative of $\mathcal{I}$. All the operators that we use on interpretations are independent from the choice of the representative. Therefore, we can define any operator on $\mathbb{I}_{\mathbb{C}}$ in terms of its counterpart defined on $\mathbb{C}$, independently from the choice of the representative. All the definitions are independent from the choice of the syntactic object. To simplify the notation, we denote the corresponding operators on $\mathbb{I}_{\mathbb{C}}$ and $\mathbb{C}$ by the same name.

Our semantic definitions use two basic operations on equation sets.

1. Let $\mathcal{E}_1$, $\mathcal{E}_2 \in \mathbb{E}$. Then $\mathcal{E}_1 \otimes^{\mathbf{x}}_{\mathbf{y}} \mathcal{E}_2$ denotes the set

$$\big\{\mathsf{E} \mid \mathsf{E}_1 \in \mathcal{E}_1, \mathsf{E}'_2 \text{ is a renamed version of } \mathsf{E}_2 \in \mathcal{E}_2, \text{ obtained by renaming apart (from } \mathsf{E}_1, \mathbf{x} \text{ and } \mathbf{y}) \text{ all the variables in } (\mathit{var}(\mathsf{E}_1) \cup \{\mathbf{x}\}) \setminus \{\mathbf{y}\}, \mathsf{E}_1 \cup \mathsf{E}'_2 \text{ is solvable and } \mathsf{E} \text{ is a solved form of } \mathsf{E}_1 \cup \mathsf{E}'_2\big\}.$$

   We assume $\otimes^{\mathbf{x}}_{\mathbf{y}}$ to be left-associative. In the following to simplify the notation, for any $\mathcal{E}_1, \mathcal{E}_2$, we denote $\mathcal{E}_1 \otimes^{\mathbf{x}}_{\mathbf{V}} \mathcal{E}_2$ simply by $\mathcal{E}_1 \otimes \mathcal{E}_2$.

2. Let E be an equation set. $\mathsf{E}|_{\mathsf{x}}$ denotes the set $\{\mathsf{x} = \mathsf{t} \in \mathsf{E} \mid \mathsf{x} \text{ occurs in } \mathbf{x}\}$. The $|_{\mathsf{x}}$ operation can be trivially extended to any $\mathcal{E} \in \mathbb{E}$ in the following way.

$$\mathcal{E}|_{\mathsf{x}} := \big\{\mathsf{E}|_{\mathsf{x}} \mid \mathsf{E} \text{ is a solved form of } \mathsf{E}' \in \mathcal{E}\big\}.$$

Finally note that, for any choice of solved form, the results are equivalent modulo variance.

All the proofs of the results of this chapter can be found in Section 4.6.

4.1.1 Denotational Semantics

The denotational semantics $\mathcal{F}[\![\mathsf{P}]\!]$ of a program P is defined as the least fixpoint of the following monotonic immediate consequence operator $\mathcal{P}[\![\mathsf{P}]\!] : \mathbb{I}_{\mathbb{C}} \to \mathbb{I}_{\mathbb{C}}$, i.e., $\mathcal{F}[\![\mathsf{P}]\!] := \mathit{lfp}\, \mathcal{P}[\![\mathsf{P}]\!]$.

$$\mathcal{P}[\![\mathsf{P}]\!](\mathcal{I}) := \lambda \mathsf{p}(\mathbf{x}). \bigcup \big\{\mathcal{E} \mid \mathsf{c} = \mathsf{p}(\mathbf{x}) \leftarrow \mathsf{E}, \mathsf{A}_1, \ldots, \mathsf{A}_n \text{ is a renamed clause of } \mathsf{P}, \mathbf{z} = \mathit{var}(\mathsf{c}), \text{ for } \mathsf{i} \in [1, n], \mathbf{y}_{\mathsf{i}} = \mathit{var}(\mathsf{A}_{\mathsf{i}}), \mathcal{E} = (\{\mathsf{E}\} \otimes^{\mathbf{z}}_{\mathbf{y}_1} \mathcal{I}(\mathsf{A}_1) \otimes^{\mathbf{z}}_{\mathbf{y}_2} \cdots \otimes^{\mathbf{z}}_{\mathbf{y}_n} \mathcal{I}(\mathsf{A}_n))|_{\mathbf{x}}\big\}.$$

This operator is isomorphic to the immediate consequence operator of the s-semantics. Indeed it is easy to see that it is just a matter of representation. In the s-semantics case, the substitution is simply applied to the pure atom, while in our case, given the pure atom, the corresponding equational version of the substitution is returned.

Example 4.1.1

Consider the program P of Figure 4.1, which is the well-known *sum* program. Its denotational semantics is

$$\mathcal{F}[\![\mathsf{P}]\!] = \mathsf{sum}(x,y,z) \mapsto \big\{\{x=0, z=y\}, \{x=s(0), z=s(y)\}, \ldots, \\ \{x=s^n(0), z=s^n(y)\}, \ldots\big\}.$$

Note that the fixpoint $\mathcal{F}[\![\mathsf{P}]\!]$ can be reached only at ω since the program computes infinite (different) solutions.

4.1.2 Operational Semantics

Definite clauses have a natural computational reading based on the resolution procedure. The specific resolution strategy, called *(equational) SLD-derivation*[2], is described in the following. For the sake of simplicity we present it using the leftmost selection rule, but all the results hold for any selection rule, since computed answers are independent from the selection rule. Let $\mathbf{G} := \mathcal{E}, p(\mathbf{x}), \mathbf{B}$ be an (equational) goal and $c := p(\mathbf{x}) \leftarrow E, \mathbf{B}'$ be a clause, such that $var(c) \cap var(\mathbf{G}) = \mathbf{x}$ and $\mathcal{E} \otimes \{E\} \neq \bot_{\mathbb{E}}$. Then we have an *(equational) derivation step*

$$\mathbf{G} \xrightarrow[c]{eq} \mathcal{E} \otimes \{E\}, \mathbf{B}', \mathbf{B}.$$

A derivation of a goal $\mathbf{G}$ in a program P, $\mathbf{G} \xrightarrow[P]{eq}{}^{*} \mathbf{G}'$ is a finite sequence of derivation steps $\mathbf{G} \xrightarrow[c_1]{eq} \cdots \xrightarrow[c_k]{eq} \mathbf{G}'$, where $c_1, \ldots, c_k$ are renamed clauses of P such that $\forall i, j \in [1,k]$, $j < i$. $c_i = p(\mathbf{x}_i) \leftarrow \mathbf{G}_i$ and $var(c_i) \cap (var(\mathbf{G}) \cup var(c_j)) = \mathbf{x}_i$.

The *behavior* (set of computed answers) of the query $\mathcal{E}, \mathbf{B}$ *in* P is

$$\mathcal{B}[\![\mathcal{E}, \mathbf{B} \textit{ in } \mathsf{P}]\!] := \bigcup \big\{\mathcal{E}'|_{var(\mathbf{B})} \mid \mathcal{E}, \mathbf{B} \xrightarrow[P]{eq}{}^{*} \mathcal{E}', \square\big\}.$$

The *top-down (goal-independent) denotation* of a program P is the interpretation obtained by collecting the behaviors for all pure atomic goals, i.e.,

$$\mathcal{O}[\![\mathsf{P}]\!] := (\lambda p(\mathbf{x}).\ \mathcal{B}[\![p(\mathbf{x}) \textit{ in } \mathsf{P}]\!]) \big/_{\equiv_{\mathbb{C}}}.$$

The main properties of the concrete semantics are summarized by the following theorem.

Theorem 4.1.2 *Let* P *be a program,* $\mathbf{G} := \mathcal{E}, A_1, \ldots, A_n$ *be a goal,* $\mathbf{y}_i := var(A_i)$ *(for* $i \in [1,n]$*) and* $\mathbf{z} := var(A_1, \ldots, A_n)$*. Then*

[2]This equational *SLD*-derivation is an extension of the equational version of an *SLD*-derivation introduced in Section 1.3.3.

```
c1: sum(0,X,X).
c2: sum(s(X),Y,s(Z)) :- sum(X,Y,Z).
```

Figure 4.1: The sum program of Examples 4.1.1 and 5.5.4

1. $\mathcal{B}[\![\mathbf{G} \textit{ in } P]\!] = (\mathcal{E} \otimes^z_{y_1} \mathcal{O}[\![P]\!](A_1) \otimes^z_{y_2} \cdots \otimes^z_{y_n} \mathcal{O}[\![P]\!](A_n))|_z$,
2. $\mathcal{P}[\![P]\!]$ *is continuous on* $\mathbb{I}_{\mathbb{C}}$ *(and therefore* $\mathcal{F}[\![P]\!] = \mathcal{P}[\![P]\!]\uparrow\omega$*)*,
3. $\mathcal{F}[\![P]\!] = \mathcal{O}[\![P]\!]$.

The property expressed in Point 1 is usually called *AND-compositionality* and is sometimes referred to as *condensing* in the program analysis field. It essentially shows that the behavior of any (conjunctive) goal can be derived from the goal-independent denotation $\mathcal{O}[\![P]\!]$, i.e., from the behaviors of (finitely many) pure atomic goals. It is the property which allows us to take $\mathcal{O}[\![P]\!]$ as *the* semantics of a program, without being concerned with the behaviors for all possible goals. The validity of the condensing property is relevant to diagnosis, since it allows us to compare the expected and actual goal-independent behaviors, i.e., the specification is the intended $\mathcal{O}[\![P]\!]$.

The properties expressed in Points 2 and 3 show that the goal-independent denotation can equivalently be computed in a bottom-up way as least fixpoint of the immediate consequence operator. This is again very important for the diagnosis problem, since the basic diagnosis algorithm requires the existence of an immediate consequence operator (see Section 5.3). Moreover, the equivalence of the top-down and the bottom-up definitions of the denotation will allow us to define (see Section 5.2.2) equivalent top-down diagnosis algorithms, based on oracle simulation.

Example 4.1.3

We show now an example to stress the differences between this collecting semantics and the one of Chapter 2. Thus we would like to consider again the ancestor program of Figure 2.10, but now we have to take into account its equational version P in Figure 4.2.

Let us consider the goal $\mathbf{G} := anc(x, y)$ (which is a shorthand for $\{\emptyset\}, anc(x, y)$) and the clause $c := anc(x, y) \leftarrow \{u = x, z = y, v = w\}, par(u, v), anc(w, z)$. Since $var(c) \cap var(\mathbf{G}) = \{x, y\}$ and $\emptyset \otimes \{u = x, z = y, v = w\} = \{u = x, z = y, v = w\}$ we can do the derivation step

$$anc(x, y) \xrightarrow[c]{eq} \{u = x, z = y, v = w\}, par(u, v), anc(w, z).$$

$$\mathsf{anc}(X,Y) \leftarrow \{X = V, Y = Z\}, \mathsf{par}(V,Z)$$
$$\mathsf{anc}(X,Y) \leftarrow \{X = V, Y = Z, U = W\}, \mathsf{par}(V,W), \mathsf{anc}(U,Z)$$
$$\mathsf{par}(X,Y) \leftarrow \{X = \mathsf{joe}, Y = \mathsf{jay}\},$$
$$\mathsf{par}(X,Y) \leftarrow \{X = \mathsf{jay}, Y = \mathsf{tim}\},$$

Figure 4.2: The (equational version of the) ancestor program (see also Figure 2.10)

Going further with all the possible computations (which are a finite number) we obtain the refutations

$$d_1 := \mathsf{par}(u,v) \xrightarrow[\mathsf{par}(u,v)\leftarrow\{u=\mathsf{jay},v=\mathsf{tim}\},\square]{\mathsf{eq}} \{u = \mathsf{jay}, v = \mathsf{tim}\}, \square$$
$$d_2 := \mathsf{par}(u,v) \xrightarrow[c_1]{\mathsf{eq}} \{u = \mathsf{joe}, v = \mathsf{jay}\}, \square$$
$$d_3 := \mathsf{anc}(x,y) \xrightarrow[c_2]{\mathsf{eq}} \{u = x, v = y\}, \mathsf{par}(u,v) \xrightarrow[c_1]{\mathsf{eq}}$$
$$\{u = \mathsf{joe}, v = \mathsf{jay}, x = \mathsf{joe}, y = \mathsf{jay}\}, \square$$
$$d_4 := \mathsf{anc}(x,y) \xrightarrow[c_2]{\mathsf{eq}} \{u = x, v = y\}, \mathsf{par}(u,v) \xrightarrow[c_3]{\mathsf{eq}}$$
$$\{u = \mathsf{jay}, v = \mathsf{tim}, x = \mathsf{jay}, y = \mathsf{tim}\}, \square$$
$$d_5 := \mathsf{anc}(x,y) \xrightarrow[c]{\mathsf{eq}} \{u = x, z = y, v = w\}, \mathsf{par}(u,v), \mathsf{anc}(w,z) \xrightarrow[c_1]{\mathsf{eq}}$$
$$\{x = \mathsf{joe}, z = y, w = \mathsf{jay}, u = \mathsf{joe}, v = \mathsf{jay}\}, \mathsf{anc}(w,z) \xrightarrow[c_4]{\mathsf{eq}}$$
$$\{x = \mathsf{joe}, z = y, w = \mathsf{jay}, u = \mathsf{joe}, v = \mathsf{jay}, v' = \mathsf{jay}, w' = z\}, \mathsf{par}(v',w')$$
$$\xrightarrow[c_5]{\mathsf{eq}} \{x = \mathsf{joe}, y = \mathsf{tim}, w = \mathsf{jay}, u = \mathsf{joe}, v = \mathsf{jay}, v' = \mathsf{jay}, z = \mathsf{tim},$$
$$w' = \mathsf{tim}\}, \square$$

where

$$c_1 := \mathsf{par}(u,v) \leftarrow \{u = \mathsf{joe}, v = \mathsf{jay}\}, \square$$
$$c_2 := \mathsf{anc}(x,y) \leftarrow \{u = x, v = y\}, \mathsf{par}(u,v)$$
$$c_3 := \mathsf{par}(u,v) \leftarrow \{u = \mathsf{jay}, v = \mathsf{tim}\}, \square$$
$$c_4 := \mathsf{anc}(w,z) \leftarrow \{v' = w, w' = z\}, \mathsf{par}(v',w')$$
$$c_5 := \mathsf{par}(v',w') \leftarrow \{v' = \mathsf{jay}, w' = \mathsf{tim}\}, \square$$

which are graphically represented in Figure 4.3. Then, by its definition, we obtain

$$\mathcal{O}[\![P]\!] = \begin{cases} \mathsf{anc}(x,y) \mapsto \big\{\{x = \mathsf{joe}, y = \mathsf{tim}\},\ \{x = \mathsf{jay}, y = \mathsf{tim}\}, \\ \qquad\qquad \{x = \mathsf{joe}, y = \mathsf{jay}\}\big\} \\ \mathsf{par}(x,y) \mapsto \big\{\{x = \mathsf{jay}, y = \mathsf{tim}\},\ \{x = \mathsf{joe}, y = \mathsf{jay}\}\big\} \end{cases}$$

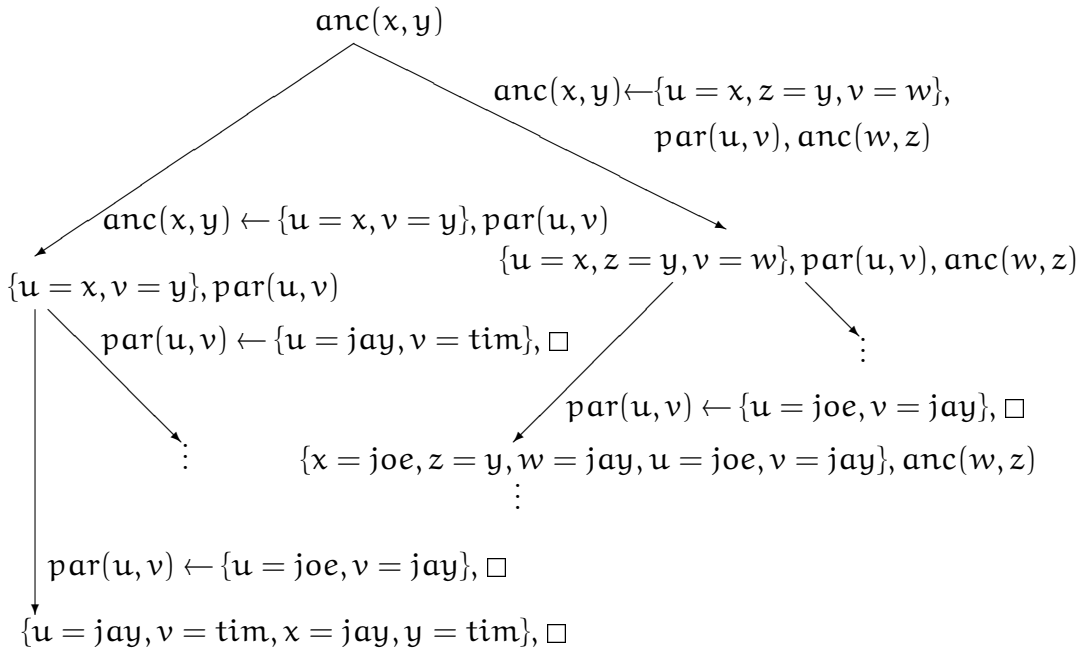

Figure 4.3: The refutations of Example 4.1.3

Moreover, by simply applying $\mathcal{P}[\![P]\!]$ to the undefined collection, we obtain

$$\mathcal{P}[\![P]\!]\uparrow_1 = par(x, y) \mapsto \big\{\{x = jay, y = tim\},\ \{x = joe, y = jay\}\big\}$$

and then, by applying $\mathcal{P}[\![P]\!]$ again, we obtain

$$\mathcal{P}[\![P]\!]\uparrow_2 = \begin{cases} anc(x, y) \mapsto \big\{\{x = joe, y = tim\},\ \{x = jay, y = tim\}, \\ \qquad\qquad \{x = joe, y = jay\}\big\} \\ par(x, y) \mapsto \big\{\{x = jay, y = tim\},\ \{x = joe, y = jay\}\big\} \end{cases}$$

which is indeed equal to $\mathcal{P}[\![P]\!]\uparrow_3$ (and $\mathcal{O}[\![P]\!]$) and then it is the least fixpoint of $\mathcal{P}[\![P]\!]$.

4.2 The Observables and the Abstract Semantics

We model the abstractions by using abstract interpretation theory [32, 34]. As already mentioned, the resulting semantic framework is a simplification of the framework in Chapter 3.

An observable property domain is a set of properties of computed answers with an ordering relation which can be viewed as an approximation structure. An observation consists in looking at a computed answer (a relevant solved form equation set), and then extracting some property (abstraction).

Definition 4.2.1 *Let* $(\mathbb{D}, \leq)$ *be a complete lattice. A function* $\alpha : \mathbb{E} \to \mathbb{D}$ *is an* observable *if it maps finite elements of* $\mathbb{E}$ *into finite elements of* $\mathbb{D}$[3] *and there exists* γ *such that*

1. $\langle \alpha, \gamma \rangle : (\mathbb{E}, \subseteq) \rightleftharpoons (\mathbb{D}, \leq)$ *is a Galois insertion,*
2. $\forall \mathcal{E}, \mathcal{E}' \in \mathbb{E}$, $\forall$ *finite* $\mathbf{x}$ *such that* $elim(\mathcal{E}) \cup elim(\mathcal{E}') \subseteq \mathbf{x} \subseteq var(\mathcal{E}) \cup var(\mathcal{E}')$, $\mathcal{E} \equiv_{\mathbf{x}} \mathcal{E}' \Longrightarrow \gamma(\alpha(\mathcal{E})) \equiv_{\mathbf{x}} \gamma(\alpha(\mathcal{E}'))$,
3. $\forall \mathcal{E} \in \mathbb{E}$, $\forall \mathsf{D} \in \mathbb{D}$ *and for any renaming*[4] ρ, $\alpha(\mathcal{E}\rho) = (\alpha(\mathcal{E}))\rho$ *and* $\gamma(\mathsf{D}\rho) = (\gamma(\mathsf{D}))\rho$.

Once we have an observable $\alpha : \mathbb{E} \to \mathbb{D}$, we want to systematically derive the abstract semantics. The idea is to define the optimal abstract versions of the various semantic operators and then check under which conditions (on the observable) we obtain the optimal abstract semantics. This will allow us to identify some interesting classes of observables.

We start by defining the optimal abstract counterparts of the basic operators defined on $\mathbb{E}$. Then, for any $\mathsf{D}, \mathsf{D}', \mathsf{D}_i \in \mathbb{D}$,

1. $\mathsf{D} \widetilde{\otimes}_{\mathbf{x}}^{\mathbf{y}} \mathsf{D}' := \alpha(\gamma(\mathsf{D}) \otimes_{\mathbf{x}}^{\mathbf{y}} \gamma(\mathsf{D}'))$, $\quad (\mathsf{D} \widetilde{\otimes} \mathsf{D}' := \alpha(\gamma(\mathsf{D}) \otimes \gamma(\mathsf{D}')))$,
2. $\mathsf{D}\tilde{|}_{\mathbf{x}} := \alpha((\gamma(\mathsf{D}))|_{\mathbf{x}})$,
3. $\widetilde{\bigcup} \{\mathsf{D}_i\}_{i \in I} := \alpha\left(\bigcup \{\gamma(\mathsf{D}_i)\}_{i \in I}\right)$.

For all $\mathsf{C} \in \mathbb{C}$ and $\mathbf{G} \in \mathit{Goals}$, let[5] $\alpha^\star(\mathsf{C}) := \lambda \mathbf{G}.\, \alpha(\mathsf{C}(\mathbf{G}))\tilde{|}_{var(\mathbf{G})}$. $\mathbb{A} := \alpha^\star(\mathbb{C}) \subseteq [\mathit{Goals} \rightharpoonup \mathbb{D}]$ is a complete lattice of partial functions from *Goals* to the abstract domain $\mathbb{D}$, ordered by the trivial extension to functions of $\leq$, which, by abuse of notation, is denoted also by $\leq$. In the following we denote by $\widetilde{\bigsqcup}$ and $\widetilde{\bigsqcap}$ the *lub* and *glb* of $\mathbb{A}$ respectively. Moreover we call $\mathbb{A}$-collection any element of $\mathbb{A}$.

The insertion $\langle \alpha, \gamma \rangle$ can be lifted to $\langle \alpha^\star, \gamma^\star \rangle : \mathbb{C} \rightleftharpoons \mathbb{A}$ by defining, for all $\mathsf{S} \in \mathbb{A}$ and $\mathbf{G} \in \mathit{Goals}$, $\gamma^\star(\mathsf{S}) := \lambda \mathbf{G}.\, (\gamma(\mathsf{S}(\mathbf{G})))|_{var(\mathbf{G})}$. From now on we will often abuse notation and denote $\alpha^\star$ by α (and $\gamma^\star$ by γ). Note that in the following if there exists a bijective Galois insertion between two domains, we identify them. A *pure* $\mathbb{A}$-collection is an element $\mathsf{S} \in \mathbb{A}$ which is undefined for every non-pure atom.

[3]We assume that the elements of the domain $\mathbb{D}$ can be represented by means of a syntactic expression built over the (free) variables which appear in the corresponding equations in $\mathbb{E}$. By *finite element* of $\mathbb{D}$ we mean any element which is finitely representable in the domain $\mathbb{D}$. Indeed the only relevant thing to our purposes is that a finite element has a finite number of free variables.

[4]Note that the renaming operation has to be applied to the syntactic representation of the abstract object.

[5]Remember that if $\mathsf{C}(\mathbf{G})$ is undefined then also $\alpha(\mathsf{C}(\mathbf{G}))$ is undefined.

Point 2 of Definition 4.2.1 states that the observation does not depend on the choice of the hidden variable names and on the choice of the solved form of the equations used in the computation. Hence, we can define an *abstract enhanced variance relation* $\equiv_{\mathbb{A}}$ on pure $\mathbb{A}$-collections as follows: for any pure $\mathbb{A}$-collection S, S', $S \equiv_{\mathbb{A}} S' \iff \gamma(S) \equiv_{\mathbb{C}} \gamma(S')$. An $\mathbb{A}$-*interpretation* is a pure $\mathbb{A}$-collection modulo $\equiv_{\mathbb{A}}$. Since (by Point 2 of Definition 4.2.1) $D \equiv_{\mathbb{C}} D'$ implies $\alpha(D) \equiv_{\mathbb{A}} \alpha(D')$, then the $\mathbb{A}$-interpretation $\alpha(I)$ is well defined, for any $\mathbb{C}$-interpretation I. Furthermore, Point 3 of Definition 4.2.1 states that the observation does not depend upon a particular choice of variable names. We denote by $(\mathbb{I}_{\mathbb{A}}, \leq)$ the complete lattice of $\mathbb{A}$-interpretations. Any $\mathbb{A}$-interpretation $\mathcal{I}_\alpha$ of $\mathbb{I}_{\mathbb{A}}$ is implicitly considered also as an arbitrary $\mathbb{A}$-collection obtained by choosing an arbitrary representative of $\mathcal{I}_\alpha$. All the semantic operators that we use on $\mathbb{A}$-interpretations are independent from the choice of the representative. Therefore, we can define any operator on $\mathbb{I}_{\mathbb{A}}$ in terms of its counterpart defined on $\mathbb{A}$, independently from the choice of the representative.

Once we have the optimal abstract operators, we can define the corresponding abstract semantics, obtained from the denotational and operational semantics of computed answers, by replacing the concrete semantic operators by their optimal abstract versions.

4.2.1 Abstract Denotational Semantics

An *abstract goal* is an object of the form $D, \mathbf{B}$ where $\mathbf{B}$ is a pure goal and $D \in \mathbb{D}$ is a finite abstract constraint, such that $D \leq D\tilde{|}_{var(\mathbf{B})}$ (i.e., D is relevant for $\mathbf{B}$). $\alpha(\{\emptyset\}), \mathbf{B}$ will be simply denoted by $\mathbf{B}$.

The abstract denotational semantics of a program P is defined as the least fixpoint of the following monotonic "abstract immediate consequence" operator $\mathcal{P}_\alpha[\![P]\!] : \mathbb{I}_{\mathbb{A}} \to \mathbb{I}_{\mathbb{A}}$, i.e., $\mathcal{F}_\alpha[\![P]\!] := \mathit{lfp}\, \mathcal{P}_\alpha[\![P]\!]$.

$$\begin{aligned}\mathcal{P}_\alpha[\![P]\!](\mathcal{I}_\alpha) := \lambda p(\mathbf{x}).\ \widetilde{\bigcup}\big\{ & D \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed} \\ & \text{clause of } P,\ \mathbf{z} = \mathit{var}(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = \mathit{var}(A_i), \\ & D = (\alpha(\{E\}) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_1} \mathcal{I}_\alpha(A_1) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_2} \cdots \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_n} \mathcal{I}_\alpha(A_n))\tilde{|}_{\mathbf{x}}\big\}.\end{aligned}$$

4.2.2 Abstract Operational Semantics

Let $\mathbf{G} := D, p(\mathbf{x}), \mathbf{B}$ be an abstract goal and $c := p(\mathbf{x}) \leftarrow E, \mathbf{B}'$ be a clause, such that $\mathit{var}(c) \cap \mathit{var}(\mathbf{G}) = \mathbf{x}$ and $D \widetilde{\otimes} \alpha(\{E\})$ is a finite element of $\mathbb{D}$ different from $\bot_{\mathbb{D}}$.

Then we have an *abstract derivation step*[6]

$$\mathbf{G} \xrightarrow[c]{\alpha-eq} D \widetilde{\otimes} \alpha(\{E\}), \mathbf{B}', \mathbf{B}.$$

An *abstract derivation* of a goal $\mathbf{G}$ in a program P, $\mathbf{G} \xrightarrow[P]{\alpha-eq}{}^{*} \mathbf{G}'$ is a finite sequence of derivation steps $\mathbf{G} \xrightarrow[c_1]{\alpha-eq} \cdots \xrightarrow[c_k]{\alpha-eq} \mathbf{G}'$, where $c_1, \ldots, c_k$ are renamed clauses of P such that $\forall i, j \in [1, k]$, $j < i$. $c_i = p(\mathbf{x}_i) \leftarrow \mathbf{G}_i$ and $var(c_i) \cap (var(\mathbf{G}) \cup var(c_j)) = \mathbf{x}_i$.

The *abstract behavior* of the abstract query $D, \mathbf{B}$ *in* P is

$$\mathcal{B}_\alpha[\![D, \mathbf{B} \textit{ in } P]\!] := \widetilde{\bigcup} \{ D'\tilde{|}_{var(\mathbf{B})} \mid D, \mathbf{B} \xrightarrow[P]{\alpha-eq}{}^{*} D', \square \}$$

and the *abstract top-down denotation* of a program P is the $\mathbb{A}$-interpretation

$$\mathcal{O}_\alpha[\![P]\!] := (\lambda p(\mathbf{x}).\ \mathcal{B}_\alpha[\![p(\mathbf{x}) \textit{ in } P]\!]) \big/_{\equiv_{\mathbb{A}}}.$$

As already discussed at the end of Section 4.1, the semantic properties which are relevant to the diagnosis problem are the condensing property and the equivalence between the top-down and the bottom-up definitions of the denotation. An additional relevant issue is precision, i.e., the relation between the abstract semantics and the abstraction of the concrete semantics. We will then identify two interesting classes of observables (complete and approximate). All the observables in both classes are condensing and lead to equivalent top-down and bottom-up definitions. Complete observables are also precise, i.e., $\alpha(\mathcal{F}[\![P]\!]) = \mathcal{F}_\alpha[\![P]\!]$, while approximate observables (intended to be used for program analysis) lead to approximated abstract semantics, i.e., $\alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{F}_\alpha[\![P]\!]$. Note that there exist observables, which are interesting for program analysis and which are not even approximate (for example, the domain of groundness dependencies $\mathcal{DEF}$ [8] is not condensing). However, as already mentioned in the introduction, there is evidence that non-condensing observables can systematically be transformed into condensing observables, by using domain refinement operators.

4.3 Complete Observables

First of all recall that any observable is precise w.r.t. the union operation since, for any Galois insertion,

$$\alpha(\bigcup_{i \in I} \mathcal{E}_i) = \alpha(\bigcup_{i \in I} (\gamma \circ \alpha)(\mathcal{E}_i)) = \widetilde{\bigcup_{i \in I}} \alpha(\mathcal{E}_i). \tag{4.3.1}$$

[6]Note that in the following we will guarantee that the $\widetilde{\otimes}$ operation will map finite arguments to finite results. For complete observables (Section 4.3) this will be consequence of the precision, while for approximate observables this will be consequence of the Noetherianity of the domain.

Definition 4.3.1 *Let $\alpha : \mathbb{E} \to \mathbb{D}$ be an observable. Then α is a* complete observable *if*

1. $\alpha(\mathcal{E}_1 \otimes_{\mathbf{x}}^{\mathbf{y}} \mathcal{E}_2) = \alpha((\gamma \circ \alpha)\mathcal{E}_1 \otimes_{\mathbf{x}}^{\mathbf{y}} (\gamma \circ \alpha)\mathcal{E}_2)$,
2. $\alpha(\mathcal{E}|_{\mathbf{x}}) = \alpha(((\gamma \circ \alpha)\mathcal{E})|_{\mathbf{x}})$.

Note that, for any finite $\mathsf{D}, \mathsf{D}' \in \mathbb{D}$, $\mathsf{D} \widetilde{\otimes} \mathsf{D}'$ is finite since (by additivity of α) there exist two finite elements $\mathcal{E}, \mathcal{E}' \in \mathbb{E}$ such that $\mathsf{D}, \mathsf{D}' = \alpha(\mathcal{E}), \alpha(\mathcal{E}')$ and then (by Point 1 of Definition 4.3.1) $\alpha(\gamma\alpha(\mathcal{E}) \otimes \gamma\alpha(\mathcal{E}'))$ is finite.

Theorem 4.3.2 *Let $\alpha : \mathbb{E} \to \mathbb{D}$ be a complete observable, $\mathbf{G} := \mathsf{D}, \mathsf{A}_1, \ldots, \mathsf{A}_n$ be an abstract goal, P be a program, $\mathbf{z} := var(\mathsf{A}_1, \ldots, \mathsf{A}_n)$ and $\mathbf{y}_i := var(\mathsf{A}_i)$ (for $i \in [1, n]$). Then*

1. $\alpha(\mathcal{O}[\![\mathsf{P}]\!]) = \mathcal{O}_\alpha[\![\mathsf{P}]\!]$,
2. $\mathcal{B}_\alpha[\![\mathbf{G} \text{ in } \mathsf{P}]\!] = (\mathsf{D} \widetilde{\otimes}_{\mathbf{y}_1}^{\mathbf{z}} \mathcal{O}_\alpha[\![\mathsf{P}]\!](\mathsf{A}_1) \widetilde{\otimes}_{\mathbf{y}_2}^{\mathbf{z}} \cdots \widetilde{\otimes}_{\mathbf{y}_n}^{\mathbf{z}} \mathcal{O}_\alpha[\![\mathsf{P}]\!](\mathsf{A}_n))\tilde{|}_{\mathbf{z}}$,
3. $\alpha(\mathcal{P}[\![\mathsf{P}]\!](\mathcal{I})) = \mathcal{P}_\alpha[\![\mathsf{P}]\!](\alpha(\mathcal{I}))$,
4. $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ *is continuous on* $\mathbb{I}_{\mathbb{A}}$ *(and therefore* $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathcal{P}_\alpha[\![\mathsf{P}]\!]\uparrow\omega$*)*,
5. $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!]$,
6. $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathcal{O}_\alpha[\![\mathsf{P}]\!]$.

Theorem 4.3.2 shows that complete observables satisfy all the properties of the concrete semantics. In particular, they are condensing (Point 2) and the abstract top-down and bottom-up semantics are equivalent (Point 6). In addition, complete observables are precise (Points 1 and 5). The class of complete observables includes *computed answers* (the *Id* observable) and the observables *correct answers* and *ground instances of computed answers*, whose semantics are the atomic logical consequence semantics (c-semantics) and the least Herbrand model, respectively.

4.3.1 The Correct Answer Observable

We show how to obtain a semantics which models correct answers and is isomorphic to the atomic logical consequence semantics (c-semantics) [16, 45]. A similar construction can be used to define ground correct answers leading to the least Herbrand model semantics.

Correct answers are closed under instantiation. This property corresponds to the downward closure of the corresponding equation sets. These sets can be (efficiently) handled with maximal equation sets w.r.t. $\leq_e$.

First of all consider equation sets modulo $\approx_e$. Any $\mathcal{E} \in \mathbb{E}/_{\approx_e}$ is *non-redundant* (w.r.t. $\leq_e$) if and only if $\forall \mathsf{E}, \mathsf{E}' \in \mathcal{E}.\, \mathsf{E} \neq \mathsf{E}' \Longrightarrow \mathsf{E} \not\leq_e \mathsf{E}'$. We denote by $\lceil \cdot \rceil$ the non-redundancy operator, i.e., $\lceil \mathcal{E} \rceil := \{\mathsf{E} \mid \mathsf{E} \in \mathcal{E}, \forall \mathsf{E}' \in \mathcal{E}.\, \mathsf{E} \neq \mathsf{E}' \Longrightarrow \mathsf{E} \not\leq_e \mathsf{E}'\}$. Any downward closed equation set (correct answer set) $\mathcal{E} \in \mathbb{E}$ can be (uniquely) represented by the non-redundant equation set $\lceil \mathcal{E} \rceil$. Furthermore, since a correct answer is an instance of some computed answer, given an equation set $\mathcal{E}$ of computed answers, $\lceil \mathcal{E} \rceil$ is the representative of the corresponding equation set of correct answers.

Let $\mathbb{D}_\nu := (\lceil \mathbb{E}/_{\approx_e} \rceil, \subseteq /_{\approx_e})$ denote the complete lattice of non-redundant sets of (equivalence classes of) finite solved form equation sets, partially ordered by $\subseteq /_{\approx_e}$. Hence the *correct answer* observable $\nu : \mathbb{E} \to \mathbb{D}_\nu$ is

$$\nu(\mathcal{E}) := \lceil \mathcal{E} \rceil.$$

By applying the definitions, the abstract operators are $\mathcal{E}_1 \widetilde{\otimes} \mathcal{E}_2 = \lceil \mathcal{E}_1 \otimes \mathcal{E}_2 \rceil$, $\mathcal{E}_1 \widetilde{\otimes}^{\mathsf{y}}_{\mathsf{x}} \mathcal{E}_2 = \lceil \mathcal{E}_1 \otimes^{\mathsf{y}}_{\mathsf{x}} \mathcal{E}_2 \rceil$, $\mathcal{E}_1 \widetilde{\cup} \mathcal{E}_2 = \lceil \mathcal{E}_1 \cup \mathcal{E}_2 \rceil$ and $\widetilde{\mathcal{E}|_{\mathsf{x}}} = \lceil \mathcal{E}|_{\mathsf{x}} \rceil$. Note that $\widetilde{\otimes}$ and $\widetilde{\cup}$ are the *glb* and *lub* of $\mathbb{D}_\nu$.

ν is a complete observable and its immediate consequence operator is

$$\mathcal{P}_\nu[\![\mathsf{P}]\!](\mathcal{I}_\nu) = \lambda \mathsf{p}(\mathbf{x}).\, \lceil \mathcal{P}[\![\mathsf{P}]\!](\mathcal{I}_\nu)(\mathsf{p}(\mathbf{x}))\rceil.$$

A similar construction can be used to define ground correct answers leading to the least Herbrand model semantics.

4.4 Approximate Observables

Approximate observables are intended to model domains where we cannot require the precision property. We therefore relax the axioms of complete observables to admit non-precise $\widetilde{\otimes}$ and $\widetilde{|}$ operators. However, as already argued in the introduction, we still want to guarantee that the condensing and equivalence properties are satisfied. This can be achieved by imposing the following abstract versions of the basic properties used in Theorem 4.1.2 to guarantee condensing and equivalence in the concrete case.

Definition 4.4.1 *Let $\alpha : \mathbb{E} \to \mathbb{D}$ be an observable. Then α is an* approximate observable *if $\mathbb{D}$ is Noetherian and*

1. $\alpha(\gamma\alpha(\mathcal{E}) \otimes^{\mathsf{y}}_{\mathsf{x}} \mathcal{E}') = \alpha(\mathcal{E} \otimes^{\mathsf{y}}_{\mathsf{x}} \gamma\alpha(\mathcal{E}')) = \alpha(\gamma\alpha(\mathcal{E}) \otimes^{\mathsf{y}}_{\mathsf{x}} \gamma\alpha(\mathcal{E}'))$,
2. $\widetilde{|}$ *is additive on* $(\mathbb{D}, \leq)$,
3. $var(\alpha(\{\emptyset\})) = \emptyset$,

4. *for any* $\mathbf{x}, \mathbf{y}, \mathbf{z}$ *such that* $\mathbf{x}, \mathbf{y} \subseteq \mathbf{z}$, $(\mathsf{D}_1 \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}} \mathsf{D}_2)\tilde{|}_{\mathbf{z}} = (\mathsf{D}_1\tilde{|}_{\mathbf{z}} \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}} \mathsf{D}_2)\tilde{|}_{\mathbf{z}}$ *and* $(\mathsf{D}_1 \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}} \mathsf{D}_2)\tilde{|}_{\mathbf{x}} = (\mathsf{D}_1 \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}} \mathsf{D}_2\tilde{|}_{\mathbf{y}})\tilde{|}_{\mathbf{x}}$,

5. *let* $\mathbf{x} \subseteq \mathbf{z}$ *and* $n \geq 1$. *If, for any* $i \in [1, n]$, $var(\mathsf{D}_i) \cap (var(\mathsf{D}) \cup \mathbf{z}) \subseteq \mathbf{x}$ *then* $\mathsf{D} \widetilde{\otimes} (\mathsf{D}_1 \widetilde{\otimes} \cdots \widetilde{\otimes} \mathsf{D}_n) = \mathsf{D} \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{x}} (\mathsf{D}_1 \widetilde{\otimes} \cdots \widetilde{\otimes} \mathsf{D}_n)$,

6. *let* $\mathbf{x} \subseteq \mathbf{z}$ *and* $n \geq 1$. *If, for any* $i \in [1, n]$, $var(\mathsf{D}) \cap (var(\mathsf{D}_i) \cup \mathbf{z}) \subseteq \mathbf{x}$ *then* $(\mathsf{D}_1 \widetilde{\otimes} \cdots \widetilde{\otimes} \mathsf{D}_n) \widetilde{\otimes} \mathsf{D} = (\mathsf{D}_1 \widetilde{\otimes} \cdots \widetilde{\otimes} \mathsf{D}_n) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{x}} \mathsf{D}$.

7. $\alpha(\mathcal{E})\tilde{|}_{\mathbf{x}} = \alpha(\mathcal{E}|_{\mathbf{x}})\tilde{|}_{\mathbf{x}}$,

The Noetherianity of $\mathbb{D}$ is used to ensure the finiteness of $\widetilde{\otimes}$.

Note that, differently from the case of complete observables, the class of approximate observables is also characterized in terms of some properties of the abstract operators. An equivalent characterization, given in terms of α, γ and the concrete operators, is possible, but would be harder to understand.

Theorem 4.4.2 *Let* $\alpha : \mathbb{E} \to \mathbb{D}$ *be an approximate observable,* P *be a program,* $\mathbf{G} := \mathsf{D}, \mathsf{A}_1, \ldots, \mathsf{A}_n$ *be an abstract goal,* $\mathbf{z} := var(\mathsf{A}_1, \ldots, \mathsf{A}_n)$ *and* $\mathbf{y}_i := var(\mathsf{A}_i)$ *(for* $i \in [1, n]$*). Then*

1. $\alpha(\mathcal{O}[\![\mathsf{P}]\!]) \leq \mathcal{O}_\alpha[\![\mathsf{P}]\!]$,

2. $\mathcal{B}_\alpha[\![\mathbf{G}\ \textit{in}\ \mathsf{P}]\!] = (\mathsf{D} \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_1} \mathcal{O}_\alpha[\![\mathsf{P}]\!](\mathsf{A}_1) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_2} \cdots \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_n} \mathcal{O}_\alpha[\![\mathsf{P}]\!](\mathsf{A}_n))\tilde{|}_{\mathbf{z}}$,

3. $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ *is continuous on* $\mathbb{I}_{\mathbb{A}}$ *(and therefore* $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathcal{P}_\alpha[\![\mathsf{P}]\!]\uparrow\omega$*),*

4. $\mathcal{P}_\alpha[\![\mathsf{P}]\!] = \alpha \circ \mathcal{P}[\![\mathsf{P}]\!] \circ \gamma$,

5. $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) \leq \mathcal{F}_\alpha[\![\mathsf{P}]\!]$,

6. $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathcal{O}_\alpha[\![\mathsf{P}]\!]$.

Theorem 4.4.2 shows that approximate observables are condensing (Point 2) and that the abstract top-down and bottom-up semantics are equivalent (Point 6). However, the denotations are just correct approximations, yet they are not precise (Points 1 and 5). Note that the above characterization of approximate observables guarantees the optimality of the abstract immediate consequence operator (see Point 4).

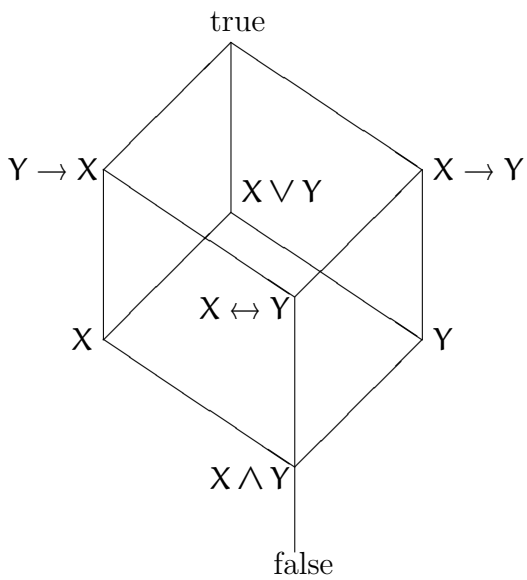

Figure 4.4: The domain $\mathcal{POS}$ for two variables x, y

4.4.1 The $\mathcal{POS}$ Observable for Groundness Dependencies of Computed Answers

We show now how to model the domain $\mathcal{POS}$, designed for the *groundness analysis* of logic programs [31, 79, 8]. $\mathcal{POS}$ is a domain of equivalence classes of propositional formulas, built using the logical connectives $\leftrightarrow$, $\wedge$ and $\vee$, and ordered by implication. The propositional formulas represent groundness dependencies among variables. The domain $\mathcal{POS}$ for two variables is shown in Figure 4.4.

First of all we have to define the abstraction $\Gamma(t)$ of a concrete term t. If $var(t) = \{x_1, \ldots, x_n\}$ then $\Gamma(t) := x_1 \wedge \cdots \wedge x_n$. The intuition is that, in order for t to be ground, all its variables $x_1, \ldots, x_n$ must be ground.

We can extend Γ to solved form equation sets to obtain abstract formulas as follows. $\Gamma(E) := \bigwedge_{x=t \in E}(x \leftrightarrow \Gamma(t))$, where $\Gamma(\emptyset) := \mathit{true}$. Abstract formulas are propositional formulas which express the *groundness dependencies* among the eliminable variables and the other variables of the concrete solved form equation set.

The *groundness dependencies of computed answer* observable $\Upsilon : \mathbb{E} \to \mathcal{POS}$ is

$$\Upsilon(\mathcal{E}) := \bigvee_{E \in \mathcal{E}} \Gamma(E),$$

where $\Upsilon(\emptyset) := \mathit{false}$. By applying the definition, the abstract operators turn out to

be

$$\begin{aligned}
&F_1 \widetilde{\otimes} F_2 = F_1 \wedge F_2 \\
&\widetilde{\bigcup}\{F_i\}_{i\in I} = \bigvee_{i\in I} F_i \\
&F\tilde{|}_x = \begin{cases} F & \text{if } var(F) \subseteq \{\mathbf{x}\} \\ (F[y \mapsto \mathit{true}] \vee F[y \mapsto \mathit{false}])\tilde{|}_x & \text{for some } y \in var(F) \setminus \{\mathbf{x}\} \end{cases}
\end{aligned}$$

where $F[y \mapsto E]$ is obtained by replacing each occurrence of the variable y in F by E. Note that the abstract notion of restriction of an abstract formula corresponds to *Schröder's elimination principle*.

The $\widetilde{\otimes}_x^y$ is performed by first renaming the second argument and then computing the conjunction. We omit its formal definition because it is not needed to define the semantics, since, in any expression we use, it collapses to $\widetilde{\otimes}$.

Υ is an approximate observable. The abstract immediate consequence operator $\mathcal{P}_\Upsilon[\![P]\!]$ is

$$\mathcal{P}_\Upsilon[\![P]\!](\mathcal{I}_\Upsilon) = \lambda p(\mathbf{x}). \bigvee_{\substack{p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \\ \text{renamed clause of } P}} (\Upsilon(\{E\}) \wedge \bigwedge_{i=1}^{n} \mathcal{I}_\Upsilon(A_i))\tilde{|}_x$$

Note that $\mathcal{P}_\Upsilon[\![P]\!]$ is defined only in terms of abstract restriction $\tilde{|}$ and of the *lub* and *glb* operators on $\mathcal{POS}$ ($\wedge$ and $\vee$ respectively).

Example 4.4.3

Consider the program P of Figure 4.5 and the groundness dependencies of computed answer observable. The abstract denotation of P is

$$\mathcal{F}_\Upsilon[\![P]\!] = \begin{cases} p(x,y) \mapsto x \vee y \\ q(x,y) \mapsto x \leftrightarrow y \\ r(x,y) \mapsto x \wedge y \end{cases}$$

which turns out to be precise since $\mathcal{F}_\Upsilon[\![P]\!] = \Upsilon(\mathcal{F}[\![P]\!])$. Note that, if we choose to represent groundness dependencies using the domain $\mathcal{DEF}$, which does not contain disjunctive formulas (and whose corresponding observable is not approximate), we would obtain a less precise abstract semantics, where the semantics of $p(x,y)$ would be *true* and the semantics of $r(x,y)$ would be $x \leftrightarrow y$.

As one might expect, we followed the same guiding line of the definition of the groundness dependencies of computed answer observable of Section 3.5.1. Note that,

```
r(X,Y) :- p(X,Y), q(X,Y).
p(a,Y).
p(X,b).
q(X,X).
```

Figure 4.5: The program of Examples 4.4.3, 5.5.2 and 5.5.8 (see also Figure 3.2)

due to the closer relationship between the application framework and the general one, the $\tau_\Gamma : \mathbb{A}_{\mathrm{ca}} \to \mathbb{A}_{\mathrm{gr}}$ abstraction and the abstraction $\lambda \mathsf{X}.\, \lambda \mathbf{G}.\, \Upsilon(\mathsf{X}(\mathbf{G}))$ are isomorphic, in the sense that, for any collection $\mathsf{X} \in \mathbb{A}_{\mathrm{ca}}$, $\tau_\Gamma(\mathsf{X}) = \Upsilon \circ \mathit{eqn} \circ \mathsf{X}$. If we lift *eqn* to $\mathbb{A}_{\mathrm{ca}}$ as $\mathit{eqn}(\mathsf{X}) := \lambda \mathbf{G}.\ \mathit{eqn}(\mathsf{X}(\mathbf{G}))$, then $\tau_\Gamma = \Upsilon \circ \mathit{eqn}$. An analogous relationship holds for the denotational semantics.

4.4.2 The *depth*(k)-answer Observable

We will show now how to approximate an infinite set of computed answers by means of a *depth*(k) cut, i.e., by cutting terms which have a depth greater than k. Terms are cut by replacing each sub-term rooted at depth k with a new variable taken from a set $\hat{\mathsf{V}}$ (disjoint from V). A *depth*(k) term represents all the terms obtained by instantiating the variables of $\hat{\mathsf{V}}$ with terms built over V.

First of all we have to define the abstraction $\mathsf{t}\lightning_{\mathsf{k}}$ as the *depth*(k) reduction of the concrete term t.

We can extend $\lightning_{\mathsf{k}}$ to solved form equation sets to obtain abstract formulas as follows. $\mathsf{E}\lightning_{\mathsf{k}} := \{\mathsf{x} = \mathsf{t}\lightning_{\mathsf{k}} \mid \mathsf{x} = \mathsf{t} \in \mathsf{E}\}$. We assume that, for any equation in E, any cut is performed by using distinct variables of $\hat{\mathsf{V}}$. We denote by $\mathbb{E}\lightning_{\mathsf{k}}$ the set of solved form equation sets with *depth*(k) terms and eliminable variables in V.

The *depth*(k) *computed answer* observable $\tau_{\mathsf{k}} : \mathbb{E} \to \mathbb{E}\lightning_{\mathsf{k}}$ is

$$\tau_{\mathsf{k}}(\mathcal{E}) := \left\{\mathsf{E}\lightning_{\mathsf{k}} \mid \mathsf{E} \in \mathcal{E}\right\}.$$

In the following we assume that, for any expression $\mathcal{F} \widetilde{\otimes}_{\mathsf{x}}^{\mathsf{y}} \mathcal{F}'$, we rename variables in V and $\hat{\mathsf{V}}$ with variables still in V and $\hat{\mathsf{V}}$ respectively. Furthermore, we consider only solved forms with eliminable variables in V. Under these assumptions, by applying the definition, the abstract operators turn out to be

$$\mathcal{F}_1 \widetilde{\otimes} \mathcal{F}_2 = (\mathcal{F}_1 \otimes \mathcal{F}_2)\lightning_{\mathsf{k}}, \qquad (\mathcal{F}_1 \widetilde{\otimes}_{\mathsf{x}}^{\mathsf{y}} \mathcal{F}_2 = (\mathcal{F}_1 \otimes_{\mathsf{x}}^{\mathsf{y}} \mathcal{F}_2)\lightning_{\mathsf{k}}),$$

$$\widetilde{\bigcup}\{\mathcal{F}_{\mathsf{i}}\}_{\mathsf{i}\in \mathrm{I}} = \bigcup_{\mathsf{i}\in \mathrm{I}} \mathcal{F}_{\mathsf{i}},$$

$$\tilde{\mathcal{F}}|_{\mathbf{x}} = \mathcal{F}|_{\mathbf{x}}.$$

τ_k is an approximate observable. The abstract immediate consequence operator $\mathcal{P}_{\tau_k}[\![P]\!]$ is

$$\begin{aligned}\mathcal{P}_{\tau_k}[\![P]\!](\mathcal{I}_{\tau_k}) = \lambda p(\mathbf{x}).\ \bigcup \big\{\mathcal{F} \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed} \\ \text{clause of } P,\ \mathbf{z} = \mathit{var}(c),\ \text{for } i \in [1,n],\ \mathbf{y}_i = \mathit{var}(A_i), \\ \mathcal{F} = ((\{E\}\lightning_k \otimes^{\mathbf{z}}_{\mathbf{y}_1} \mathcal{I}_{\tau_k}(A_1) \otimes^{\mathbf{z}}_{\mathbf{y}_2} \cdots \otimes^{\mathbf{z}}_{\mathbf{y}_n} \mathcal{I}_{\tau_k}(A_n))\lightning_k)|_{\mathbf{x}}\big\}.\end{aligned}$$

Example 4.4.4

Consider the program P of Figure 4.6 and the *depth*(2) observable. The abstract denotation of P is

$$\mathcal{F}_{\tau_k}[\![P]\!] = r(x) \mapsto \big\{\emptyset, \{x = f(z)\}, \{x = f(f(\hat{x}))\}\big\}.$$

Note that since τ_2 is approximate, the denotation of P is finite and can be computed in finite time.

Analogously to the case of $\mathcal{POS}$ (Section 4.4.1), a relation between this abstraction and the one of the general framework in Section 3.5.2 holds. Namely, for each $X \in \mathbb{A}_{ca}$, $\mathit{eqn}(\mu_k(X)(\mathbf{G})) = \tau_k(\mathit{eqn}(X(\mathbf{G})))$. An analogous relationship holds for the denotational semantics.

4.5 Discussion on the Results of the Framework for Applications

As already mentioned at the beginning of the chapter, this semantic framework was defined as the basic framework for abstract diagnosis, presented in Chapter 5. In abstract diagnosis the program properties we are interested in are operational properties and not necessarily declarative properties. This means that we cannot base it on the declarative semantics, since this semantics is too abstract to allow us to reason about some operational properties, such as variable groundness.

The main novelty of this framework is in the formulation of the observable τ (which is an abstraction of computed answers) and that the axioms to classify the observables are reformulated in terms of simpler axioms on τ. Moreover the abstract operational semantics and the abstract denotational semantics are equivalent also for approximate observables. Some preliminary versions of abstract diagnosis [25, 27] were based on the semantic framework of Part I. However in abstract diagnosis we are only concerned with properties which are abstractions of the computed answer

```
r(X).
r(f(X)) :- r(X).
```

Figure 4.6: The program of Example 4.4.4

semantics. Furthermore we are interested in equivalent top-down and bottom-up semantics, since this will allow us to define equivalent top-down and bottom-up diagnosis algorithms. Hence this framework is best suited for the needs of abstract diagnosis.

The equivalence of the abstract operational semantics and the abstract denotational semantics requires the observable to be condensing. This property is needed in abstract diagnosis where the specification is a post-condition describing a (goal-independent) property of a set of procedure declarations. It is worth noting that the observables corresponding to the declarative semantics are condensing and that the declarative semantics do indeed characterize procedure declarations. Note also that several observables used in program analysis (for mode, type and groundness analysis) are also condensing and that a non-condensing observable can systematically be transformed into a (more concrete) condensing observable, by using domain refinement operators (see, for example, how the condensing domain $\mathcal{POS}$ for groundness analysis can be derived [89] from the non-condensing domain $\mathcal{DEF}$).

As expected from their definitions, the difference between complete and approximate observables is related to precision. Namely, the abstract semantics coincides with the abstraction of the collecting semantics, in the case of complete observables, while it is just a correct approximation, in the case of approximate observables. On the other side, approximate observables correspond to Noetherian domains. Hence their abstract semantics is finite. Several useful properties are modeled by approximate observables or by observables which can be systematically refined to approximate observables. This is interesting because finite specifications lead to the systematic derivation of the diagnosis algorithms from the underlying theory with *no need for symptom detection*.

4.6 Proofs of the Chapter

First of all observe that the operations $\otimes$, $\otimes^{\mathbf{x}}_{\mathbf{y}}$ and $|$ satisfy the following properties. These properties are the ones we use to define approximate observables, and will allow us to reuse the proof of the theorems for complete observables in the

corresponding theorems for approximate observables. Let $\mathcal{E}, \mathcal{E}_i \in \mathbb{E}$.

$$var(\{\emptyset\}) = \emptyset. \tag{4.6.1}$$

$\otimes$, $\otimes^{x}_{y}$ and $|$ are additive on $\mathbb{E}$. $\otimes$ is associative, while $\otimes^{x}_{y}$ is associative in the following sense. If $\{\mathbf{y}\} \subseteq \{\mathbf{z}'\} \subseteq \{\mathbf{z}\}$ then

$$(\mathcal{E}_1 \otimes^{z}_{z'} \mathcal{E}_2) \otimes^{z}_{y} \mathcal{E}_3 = \mathcal{E}_1 \otimes^{z}_{z'} (\mathcal{E}_2 \otimes^{z'}_{y} \mathcal{E}_3). \tag{4.6.2}$$

For any $\mathbf{x}, \mathbf{y}, \mathbf{z}$ and $\mathbf{v}$,

$$(\mathcal{E}_1 \otimes^{y}_{x} \{\emptyset\}) \otimes^{v}_{z} \mathcal{E}_2 = \mathcal{E}_1 \otimes^{v}_{z} \mathcal{E}_2, \quad (\{\emptyset\} \otimes \mathcal{E}_1)|_x = \mathcal{E}_1|_x, \tag{4.6.3}$$

$$(\{\emptyset\} \otimes \mathcal{E}_1) \otimes^{v}_{z} \mathcal{E}_2 = (\mathcal{E}_1 \otimes \{\emptyset\}) \otimes^{v}_{z} \mathcal{E}_2 = \mathcal{E}_1 \otimes^{v}_{z} \mathcal{E}_2. \tag{4.6.4}$$

For any $\mathbf{x}, \mathbf{y}, \mathbf{z}$ such that $\mathbf{x}, \mathbf{y} \subseteq \mathbf{z}$,

$$\begin{aligned} &(\mathcal{E}_1 \otimes^{z}_{y} \mathcal{E}_2)|_z = (\mathcal{E}_1|_z \otimes^{z}_{y} \mathcal{E}_2)|_z \text{ and} \\ &(\mathcal{E}_1 \otimes^{z}_{y} \mathcal{E}_2)|_x = (\mathcal{E}_1 \otimes^{z}_{y} \mathcal{E}_2|_y)|_x. \end{aligned} \tag{4.6.5}$$

Let $\mathbf{x} \subseteq \mathbf{z}$ and $n \geq 1$. If for any $i \in [1, n]$. $var(\mathcal{E}_i) \cap (var(\mathcal{E}) \cup \mathbf{z}) \subseteq \mathbf{x}$ then

$$\mathcal{E} \otimes (\mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n) = \mathcal{E} \otimes^{z}_{x} (\mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n) \tag{4.6.6}$$

and if for any $i \in [1, n]$. $var(\mathcal{E}) \cap (var(\mathcal{E}_i) \cup \mathbf{z}) \subseteq \mathbf{x}$ then

$$(\mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n) \otimes \mathcal{E} = (\mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n) \otimes^{z}_{x} \mathcal{E}. \tag{4.6.7}$$

Now, we can prove the theorems.

Proof of Theorem 4.1.2.

Point 1 We prove the two inclusions separately.

$\subseteq$ The proof is by induction on the number n of atoms in $\mathcal{E}, A_1, \ldots, A_n$.

$n = 1$ Assume that $E \in \mathcal{B}[\![\mathcal{E}, p(\mathbf{x}) \textit{ in } P]\!]$. Then, by definition of $\mathcal{B}[\![\cdot]\!]$, there exists a derivation of length k

$$\mathcal{E}, p(\mathbf{x}) \xrightarrow[P]{eq} \mathcal{E} \otimes \mathcal{E}_1, \mathbf{B}_1 \xrightarrow[P]{eq} \cdots \xrightarrow[P]{eq} ((\mathcal{E} \otimes \mathcal{E}_1) \cdots) \otimes \mathcal{E}_k, \square$$

such that $E \in (((\mathcal{E} \otimes \mathcal{E}_1) \cdots) \otimes \mathcal{E}_k)|_x$. By associativity of $\otimes$ and by (4.6.4), $E \in (\mathcal{E} \otimes \mathcal{E}')|_x = (\mathcal{E} \otimes (\{\emptyset\} \otimes \mathcal{E}'))|_x$, where $\mathcal{E}' = \mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_k$. Moreover, by definition of derivation, for any $i \in [1, k]$. $var(\mathcal{E}_i) \cap var(\mathcal{E}) \subseteq \{\mathbf{x}\}$ and therefore by (4.6.6), $\mathcal{E} \otimes (\{\emptyset\} \otimes \mathcal{E}') = \mathcal{E} \otimes^{x}_{x} (\{\emptyset\} \otimes \mathcal{E}')$. Then, by the previous results and by (4.6.5), $E \in (\mathcal{E} \otimes^{x}_{x} (\{\emptyset\} \otimes \mathcal{E}')|_x)|_x$. Moreover, by definition of derivation, by (4.6.1) and by associativity of $\otimes$, $\{\emptyset\}, p(\mathbf{x}) \xrightarrow[P]{eq}{}^{*} \{\emptyset\} \otimes \mathcal{E}', \square$ and therefore, by definition of $\mathcal{O}[\![\cdot]\!]$, $(\{\emptyset\} \otimes \mathcal{E}')|_x \in \mathcal{O}[\![P]\!](p(\mathbf{x}))$.

These results, together with the additivity of $\otimes^{x}_{x}$, imply that $E \in (\mathcal{E} \otimes^{x}_{x} \mathcal{O}[\![P]\!](p(\mathbf{x})))|_x$ and complete the proof of the base case.

$\underline{n > 1}$ To simplify the notation, let us denote by z, z' and y_i, for $i \in [1, n]$, the variables $var(A_1, \ldots, A_n)$, $var(A_1, \ldots, A_{n-1})$ and $var(A_i)$, respectively. Assume that $E \in \mathcal{B}[\![\mathcal{E}, A_1, \ldots, A_n \ in \ P]\!]$. By definition of $\mathcal{B}[\![\cdot]\!]$, there exists a derivation $\mathcal{E}, A_1, \ldots, A_n \xrightarrow[P]{eq}{}^* \bar{\mathcal{E}}, \square$ such that $E \in \bar{\mathcal{E}}|_z$. Then, by definition of derivation, by (4.6.1), (4.6.4) and by associativity of $\otimes$, there exist two derivations $\{\emptyset\}, A_1, \ldots, A_{n-1} \xrightarrow[P]{eq}{}^* \{\emptyset\} \otimes \mathcal{E}', \square$ and $\{\emptyset\}, A_n \xrightarrow[P]{eq}{}^* \{\emptyset\} \otimes \mathcal{E}'', \square$ such that

$$\bar{\mathcal{E}} = \mathcal{E} \otimes \mathcal{E}' \otimes \mathcal{E}'' = \mathcal{E} \otimes (\{\emptyset\} \otimes \mathcal{E}') \otimes (\{\emptyset\} \otimes \mathcal{E}''), \tag{1}$$

where the last equality follows by (4.6.4). Then, by definition of $\mathcal{B}[\![\cdot]\!]$, $(\{\emptyset\} \otimes \mathcal{E}')|_{z'} \subseteq \mathcal{B}[\![A_1, \ldots, A_{n-1} \ in \ P]\!]$ and therefore, by inductive hypothesis,

$$\begin{aligned} &(\{\emptyset\} \otimes \mathcal{E}')|_{z'} \subseteq \\ &(\{\emptyset\} \otimes^{z'}_{y_1} \mathcal{O}[\![P]\!](A_1) \otimes^{z'}_{y_2} \cdots \otimes^{z'}_{y_{n-1}} \mathcal{O}[\![P]\!](A_{n-1}))|_{z'}. \end{aligned} \tag{2}$$

Moreover, by definition of $\mathcal{O}[\![\cdot]\!]$,

$$(\{\emptyset\} \otimes \mathcal{E}'')|_{y_n} \subseteq \mathcal{O}[\![P]\!](A_n). \tag{3}$$

Now observe that, by definition of derivation, by associativity of $\otimes$ and by (4.6.6), $\mathcal{E} \otimes (\{\emptyset\} \otimes \mathcal{E}') \otimes (\{\emptyset\} \otimes \mathcal{E}'') = (\mathcal{E} \otimes (\{\emptyset\} \otimes \mathcal{E}')) \otimes^z_{y_n} (\{\emptyset\} \otimes \mathcal{E}'')$ and $\mathcal{E} \otimes (\{\emptyset\} \otimes \mathcal{E}') = \mathcal{E} \otimes^z_{z'} (\{\emptyset\} \otimes \mathcal{E}')$. Then by (1), (2), (3), by using repeatedly (4.6.5) and by additivity of $\otimes$,

$$\begin{aligned} \bar{\mathcal{E}}|_z &= (\mathcal{E} \otimes^z_{z'} (\{\emptyset\} \otimes \mathcal{E}')|_{z'} \otimes^z_{y_n} (\{\emptyset\} \otimes \mathcal{E}'')|_{y_n})|_z \\ &\subseteq (\mathcal{E} \otimes^z_{z'} (\{\emptyset\} \otimes^{z'}_{y_1} \mathcal{O}[\![P]\!](A_1) \cdots)|_{z'} \otimes^z_{y_n} \mathcal{O}[\![P]\!](A_n))|_z \\ &= (\mathcal{E} \otimes^z_{z'} \{\emptyset\} \otimes^{z'}_{y_1} \mathcal{O}[\![P]\!](A_1) \otimes^z_{y_2} \cdots \otimes^z_{y_n} \mathcal{O}[\![P]\!](A_n))|_z. \end{aligned}$$

Then, by using repeatedly (4.6.2),

$$\bar{\mathcal{E}}|_z \subseteq (\mathcal{E} \otimes^z_{z'} \{\emptyset\} \otimes^z_{y_1} \mathcal{O}[\![P]\!](A_1) \otimes^z_{y_2} \cdots \otimes^z_{y_n} \mathcal{O}[\![P]\!](A_n))|_z$$

and therefore, by (4.6.3),

$$\bar{\mathcal{E}}|_z \subseteq (\mathcal{E} \otimes^z_{y_1} \mathcal{O}[\![P]\!](A_1) \otimes^z_{y_2} \cdots \otimes^z_{y_n} \mathcal{O}[\![P]\!](A_n))|_z.$$

This completes the proof of the first inclusion.

$\supseteq$ Let us denote by z and y_i the variables $var(A_1, \ldots, A_n)$ and $var(A_i)$, for $i \in [1, n]$, respectively. Assume that $E \in (\mathcal{E} \otimes^z_{y_1} \mathcal{O}[\![P]\!](A_1) \otimes^z_{y_2} \cdots \otimes^z_{y_n} \mathcal{O}[\![P]\!](A_n))|_z$.

Then, by additivity of $\otimes$, for any $i \in [1, n]$, there exists $E_i \in \mathcal{O}[\![P]\!](A_i)$ such that

$$E \in (\mathcal{E} \otimes^z_{y_1} \{E_1\} \otimes^z_{y_2} \cdots \otimes^z_{y_n} \{E_n\})|_z. \tag{4}$$

By definition of $\mathcal{O}[\![\cdot]\!]$, by (4.6.1) and by associativity of $\otimes$ this means that, for any $i \in [1, n]$, there exists a derivation $\{\emptyset\} \otimes A_i \xrightarrow[P]{eq}{}^* \{\emptyset\} \otimes \mathcal{E}_i, \Box$, such that $E_i \in (\{\emptyset\} \otimes \mathcal{E}_i)|_{y_i}$ and therefore by additivity of $\otimes$ and by (4),

$$\begin{aligned} E &\in (\mathcal{E} \otimes^z_{y_1} (\{\emptyset\} \otimes \mathcal{E}_1)|_{y_1} \otimes^z_{y_2} \cdots \otimes^z_{y_n} (\{\emptyset\} \otimes \mathcal{E}_n)|_{y_n})|_z \\ &= (\mathcal{E} \otimes^z_{y_1} \mathcal{E}_1|_{y_1} \otimes^z_{y_2} \cdots \otimes^z_{y_n} \mathcal{E}_n|_{y_n})|_z, \end{aligned} \tag{5}$$

where the last equality follows by (4.6.3). Now observe that, by definition of $\otimes^z_{y_i}$, we can assume that for any $i, j \in [1, n]$, $i \neq j$. $var(\mathcal{E}) \cap var(\mathcal{E}_i) \subseteq var(A_i)$ and $var(\mathcal{E}_i) \cap var(\mathcal{E}_j) = \emptyset$. Then, by definition of derivation and since $\otimes$ is associative,

$$\mathcal{E}, A_1, \ldots, A_n \xrightarrow[P]{eq}{}^* \mathcal{E} \otimes \mathcal{E}_1, A_2, \ldots, A_n \xrightarrow[P]{eq}{}^* \mathcal{E} \otimes \mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n, \Box$$

and therefore

$$(\mathcal{E} \otimes \mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n)|_z \subseteq \mathcal{B}[\![\mathcal{E}, A_1, \ldots, A_n \ in \ P]\!]. \tag{6}$$

By the previous observation on variables and by properties (4.6.5) and (4.6.7) we have that $(\mathcal{E} \otimes \mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n)|_z = (\mathcal{E} \otimes^z_{y_1} \mathcal{E}_1|_{y_1} \otimes^z_{y_2} \cdots \otimes^z_{y_n} \mathcal{E}_n|_{y_n})|_z$. This, together with (6) and (5), implies that $E \in \mathcal{B}[\![\mathcal{E}, A_1, \ldots, A_n \, in P]\!]$ and completes the proof.

Point 2 The proof that $\mathcal{P}[\![P]\!]$ is continuous on $\mathbb{I}_{\mathbb{C}}$ is straightforward by observing that $\otimes^z_y$ and $|$ are additive on $(\mathbb{E}, \subseteq)$.

Point 3 We prove the two inclusions separately.

$\sqsubseteq$ By definition of $\mathcal{F}[\![\cdot]\!]$ and by continuity of $\mathcal{P}[\![\cdot]\!]$, for any atom $p(\boldsymbol{x})$, $\mathcal{F}[\![P]\!](p(\boldsymbol{x})) = \bigcup_{h \geq 0} (\mathcal{P}[\![P]\!]{\uparrow}h)(p(\boldsymbol{x}))$. Then we have only to show that, for any h, $\mathcal{P}[\![P]\!]{\uparrow}h \sqsubseteq \mathcal{O}[\![P]\!]$. The proof is by induction on h.

$h = 0$ Straightforward, since $\mathcal{P}[\![P]\!]{\uparrow}0 = \bot_{\mathbb{C}} \sqsubseteq \mathcal{O}[\![P]\!]$.

$h > 0$ Assume that $E' \in (\mathcal{P}[\![P]\!]{\uparrow}h)(p(\boldsymbol{x}))$. By definition of $\mathcal{P}[\![\cdot]\!]$, there exists a renamed clause of P, $c = p(\boldsymbol{x}) \leftarrow E, A_1, \ldots, A_n$, such that $E' \in (\{E\} \otimes^z_{y_1} (\mathcal{P}[\![P]\!]{\uparrow}h-1)(A_1) \otimes^z_{y_2} \cdots \otimes^z_{y_n} (\mathcal{P}[\![P]\!]{\uparrow}h-1)(A_n))|_x$, where $\boldsymbol{z} = var(c)$ and for

$i \in [1, n]$, $\mathbf{y}_i = var(A_i)$. By additivity of $\otimes^{z}_{\mathbf{y}_i}$ and $|$, for any $i \in [1, n]$, there exists $E_i \in (\mathcal{P}[\![P]\!]\uparrow_{h-1})(A_i)$ such that

$$E' \in (\{E\} \otimes^{z}_{\mathbf{y}_1} \{E_1\} \otimes^{z}_{\mathbf{y}_2} \cdots \otimes^{z}_{\mathbf{y}_n} \{E_n\})|_{\mathbf{x}}. \tag{7}$$

Now observe that, by definition of $\otimes^{z}_{\mathbf{y}_i}$, we can assume, for any $i, j \in [1, n]$, $i \neq j$. $var(E) \cap var(E_i) \subseteq \mathbf{y}_i$ and $var(E_i) \cap var(E_j) = \emptyset$.
By inductive hypothesis, for any $i \in [1, n]$. $E_i \in \mathcal{O}[\![P]\!](A_i)$. Then, by Point 1, there exists a derivation $\{\emptyset\}, A_1, \ldots, A_n \xrightarrow[P]{eq}{}^{*} \{E''\}, \square$, such that $\{E''\}|_{z'} = (\{\emptyset\} \otimes^{z'}_{\mathbf{y}_1} \{E_1\} \otimes^{z'}_{\mathbf{y}_2} \cdots \otimes^{z'}_{\mathbf{y}_n} \{E_n\})|_{z'}$, where $\mathbf{z}' = var(A_1, \ldots, A_n)$. Thus, by using the renamed clause c, $\{\emptyset\}, p(\mathbf{x}) \xrightarrow[P]{eq} \{\emptyset\} \otimes \{E\}, A_1, \ldots, A_n$. Then, by using the previous two derivations, by our hypothesis on variables, by (4.6.4) and by associativity of $\otimes$, we obtain the derivation

$$\{\emptyset\}, p(\mathbf{x}) \xrightarrow[P]{eq} \{\emptyset\} \otimes \{E\}, A_1, \ldots, A_n \xrightarrow[P]{eq}{}^{*} \{E\} \otimes \{E''\}, \square$$

and, by definition of $\mathcal{O}[\![\cdot]\!]$,

$$(\{E\} \otimes \{E''\})|_{\mathbf{x}} \subseteq \mathcal{O}[\![P]\!](p(\mathbf{x})). \tag{8}$$

Moreover, by definition of derivation, by (4.6.6) and by (4.6.5)

$$(\{E\} \otimes \{E''\})|_{\mathbf{x}} = (\{E\} \otimes^{z}_{z'} \{E''\})|_{\mathbf{x}} = (\{E\} \otimes^{z}_{z'} \{E''\}|_{z'})|_{\mathbf{x}}.$$

Then, by definition of E'', by (4.6.2) and (4.6.3), $(\{E\} \otimes \{E''\})|_{\mathbf{x}} = (\{E\} \otimes^{z}_{\mathbf{y}_1} \{E_1\} \otimes^{z}_{\mathbf{y}_2} \cdots \otimes^{z}_{\mathbf{y}_n} \{E_n\})|_{\mathbf{x}}$ and therefore, by (7) and (8), $E' \in \mathcal{O}[\![P]\!](p(\mathbf{x}))$ and then the thesis.

$\sqsupseteq$ Assume that $E \in \mathcal{O}[\![P]\!](p(\mathbf{x}))$. By definition of $\mathcal{O}[\![\cdot]\!]$ there exists a derivation $\{\emptyset\}, p(\mathbf{x}) \xrightarrow[P]{eq}{}^{*} \mathcal{E}, \square$ of length $h > 0$ such that $E \in \mathcal{E}|_{\mathbf{x}}$. We prove by induction on h that $E \in (\mathcal{P}[\![P]\!]\uparrow_h)(p(\mathbf{x}))$.

$h = 1$ If there exists a derivation $\{\emptyset\}, p(\mathbf{x}) \xrightarrow[P]{eq}{}^{*} \{\emptyset\} \otimes \{E'\}, \square$ of length 1 then $p(\mathbf{x}) \leftarrow E', \square$ is a renamed clause of P and hence, by definition of $\mathcal{P}[\![\cdot]\!]$ and by (4.6.3), $(\{\emptyset\} \otimes \{E'\})|_{\mathbf{x}} = \{E'\}|_{\mathbf{x}} \in (\mathcal{P}[\![P]\!]\uparrow_1)(p(\mathbf{x}))$ and then the thesis.

$h > 1$ Assume that there exists a derivation

$$\{\emptyset\}, p(\mathbf{x}) \xrightarrow[P]{eq} \{\emptyset\} \otimes \{E'\}, A_1, \ldots, A_n \xrightarrow[P]{eq}{}^{*} \mathcal{E}, \square$$

of length $h > 1$ and let $\mathbf{y}_i = var(A_i)$, for any $i \in [1, n]$.

By definition, since $\otimes$ is associative and by (4.6.1), there exists a derivation $\{\emptyset\}, A_1, \ldots, A_n \xrightarrow[P]{eq}{}^* \{\emptyset\} \otimes \mathcal{E}', \square$ whose length is $h-1$. Then, by associativity of $\otimes$ and by (4.6.4),

$$\mathcal{E} = \{E'\} \otimes \mathcal{E}' = \{E'\} \otimes \{\emptyset\} \otimes \mathcal{E}'. \tag{9}$$

Moreover, since the length of the derivation $\{\emptyset\}, A_1, \ldots, A_n \xrightarrow[P]{eq}{}^* \{\emptyset\} \otimes \mathcal{E}', \square$ is $h-1$, the same argument used in the proof of Point 1 shows that for any $j \in [1, n]$, there exists a derivation $\{\emptyset\}, A_j \xrightarrow[P]{eq}{}^* \{\emptyset\} \otimes \mathcal{E}_j, \square$, whose length is $l_j \leq h-1$, such that

$$\{\emptyset\} \otimes \mathcal{E}' = \{\emptyset\} \otimes \mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n \tag{10}$$

and for any $i, j \in [1, n]$, $i \neq j$

$$(var(\{E'\}) \cup \{\mathbf{x}\}) \cap var(\mathcal{E}_i) \subseteq \mathbf{y}_i \text{ and}$$
$$(var(\mathcal{E}_i) \cup \{\mathbf{x}\}) \cap var(\mathcal{E}_j) = \emptyset.$$

The definition of $\mathcal{O}[\![\cdot]\!]$, (4.6.3), the inductive hypothesis and the monotonicity of $\mathcal{P}[\![P]\!]$ imply that $\mathcal{E}_i|_{\mathbf{y}_i} \subseteq (\mathcal{P}[\![P]\!]\uparrow_{h-1})(A_i)$, for any $i \in [1, n]$. Moreover, since by hypothesis there exists the derivation (of length 1) $p(\mathbf{x}) \xrightarrow[P]{eq} \{E'\}, A_1, \ldots, A_n$, there exists a renamed clause of P, $c = p(\mathbf{x}) \leftarrow E'', A_1, \ldots, A_n$, such that $\{E'\} = \{\emptyset\} \otimes \{E''\}$.
Then, by (4.6.4), by definition of $\mathcal{P}[\![\cdot]\!]\uparrow_h$ and by the additivity of $\otimes$, $(\{E''\} \otimes^z_{\mathbf{y}_1} \mathcal{E}_1|_{\mathbf{y}_1} \otimes^z_{\mathbf{y}_2} \cdots \otimes^z_{\mathbf{y}_n} \mathcal{E}_n|_{\mathbf{y}_n})|_{\mathbf{x}} = (\{E'\} \otimes^z_{\mathbf{y}_1} \mathcal{E}_1|_{\mathbf{y}_1} \otimes^z_{\mathbf{y}_2} \cdots \otimes^z_{\mathbf{y}_n} \mathcal{E}_n|_{\mathbf{y}_n})|_{\mathbf{x}} \subseteq (\mathcal{P}[\![P]\!]\uparrow_h)(p(\mathbf{x}))$, where $\mathbf{z} = var(c)$. Now, analogously to the proof of Point 1, by our hypothesis on variables,

$$(\{E'\} \otimes^z_{\mathbf{y}_1} \mathcal{E}_1|_{\mathbf{y}_1} \otimes^z_{\mathbf{y}_2} \cdots \otimes^z_{\mathbf{y}_n} \mathcal{E}_n|_{\mathbf{y}_n})|_{\mathbf{x}} = (\{E'\} \otimes \mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n)|_{\mathbf{x}}.$$

By these results, together with (9) and (10), $\mathcal{E}|_{\mathbf{x}} \subseteq (\mathcal{P}[\![P]\!]\uparrow_h)(p(\mathbf{x}))$. ■

Proof of Theorem 4.3.2. First of all observe that by Definition 4.3.1 and by definition of $\widetilde{\otimes}$ and $\tilde{|}$, $\widetilde{\otimes}$ is associative, $\widetilde{\otimes}$ and $\tilde{|}$ are additive and for any $\mathcal{E}, \mathcal{E}_1, \ldots, \mathcal{E}_n \in \mathbb{E}$ the following facts hold.

$$\alpha(\mathcal{E}_i|_{\mathbf{x}}) = (\alpha(\mathcal{E}_i))\tilde{|}_{\mathbf{x}} \text{ and} \tag{1}$$
$$\alpha(\mathcal{E}_1 \otimes \cdots \otimes \mathcal{E}_n) = \alpha(\mathcal{E}_1) \widetilde{\otimes} \cdots \widetilde{\otimes} \alpha(\mathcal{E}_n). \tag{2}$$

Now observe that, by definition of $\mathcal{B}[\![\cdot]\!]$, and since $|$ is additive and idempotent, for any goal $\mathcal{E}, \mathbf{G}$,

$$\mathcal{B}[\![\mathcal{E}, \mathbf{G} \textit{ in } P]\!] = (\mathcal{B}[\![\mathcal{E}, \mathbf{G} \textit{ in } P]\!])|_{var(\mathbf{G})}. \tag{3}$$

Analogously, for any $\mathcal{I} \in \mathbb{I}_{\mathbb{C}}$,

$$(\mathcal{P}[\![\mathsf{P}]\!](\mathcal{I})(\mathsf{p}(\mathbf{x})))|_{\mathbf{x}} = \mathcal{P}[\![\mathsf{P}]\!](\mathcal{I})(\mathsf{p}(\mathbf{x})). \tag{4}$$

Moreover, by (1), for any $\mathsf{C} \in \mathbb{C}$,

$$\alpha(\mathsf{C}) = \lambda\mathbf{G}.\,(\alpha(\mathsf{C}(\mathbf{G})))\tilde{|}_{var(\mathbf{G})} = \lambda\mathbf{G}.\,\alpha((\mathsf{C}(\mathbf{G}))|_{var(\mathbf{G})}). \tag{5}$$

Then, by definition of $\otimes$ and since for any $\mathsf{C} \in \mathbb{C}$ and for any goal $\mathbf{G}.\,\mathsf{C}(\mathbf{G}) \subseteq (\mathsf{C}(\mathbf{G}))|_{var(\mathbf{G})}$ and $(\mathsf{C}(\mathbf{G}))|_{var(\mathbf{G})}$ is the set of all solved forms of all the elements in $\mathsf{C}(\mathbf{G})$ (restricted to $var(\mathbf{G})$), for any $\mathcal{E} \in \mathbb{E}$

$$\mathcal{E} \otimes^{\mathbf{x}}_{\mathbf{y}} (\mathsf{C}(\mathbf{G}))|_{var(\mathbf{G})} = \mathcal{E} \otimes^{\mathbf{x}}_{\mathbf{y}} \mathsf{C}(\mathbf{G}). \tag{6}$$

Then, by using (2), we can prove

$$\mathsf{D} \tilde{\otimes}^{\mathbf{x}}_{\mathbf{y}} (\alpha(\mathsf{C})(\mathbf{G})) = \mathsf{D} \tilde{\otimes}^{\mathbf{x}}_{\mathbf{y}} \alpha(\mathsf{C}(\mathbf{G})). \tag{7}$$

Moreover, by definition of Galois insertion, for any $\mathsf{C}_i \in \mathbb{C}$,

$$\alpha(\bigsqcup_{i\in I} \mathsf{C}_i) = \alpha(\bigsqcup_{i\in I}((\gamma \circ \alpha)(\mathsf{C}_i))) = \widetilde{\bigsqcup_{i\in I}} \alpha(\mathsf{C}_i). \tag{8}$$

Now we can prove the theorem.

Point 1 First of all observe that, since $\alpha : \mathbb{E} \to \mathbb{D}$ is a complete observable, then it is a congruence w.r.t. renaming. Then (by using an inductive argument), from any derivation $\mathcal{E}, \mathbf{G} \xrightarrow[\mathsf{P}]{\mathrm{eq}} \mathcal{E}', \square$ we can build (a variant of) an abstract derivation $\alpha(\mathcal{E}), \mathbf{G} \xrightarrow[\mathsf{P}]{\alpha-\mathrm{eq}} \alpha(\mathcal{E}'), \square$ which uses suitable variants of all the clauses in the concrete derivation. Similarly for any $\mathsf{D}' \in \mathbb{D}$ there exists $\mathcal{E}' \in \mathbb{E}$ such that $\mathsf{D}' = \alpha(\mathcal{E}')$ and from $\alpha(\mathcal{E}), \mathbf{G} \xrightarrow[\mathsf{P}]{\alpha-\mathrm{eq}} \mathsf{D}', \mathbf{B}$ we can build (a variant of) the derivation $\mathcal{E}, \mathbf{G} \xrightarrow[\mathsf{P}]{\mathrm{eq}} \mathcal{E}', \mathbf{B}$. Since $\mathcal{B}[\![\cdot]\!]$ (and $\mathcal{B}_\alpha[\![\cdot]\!]$) collects any variant of the (abstract) derivations then the following facts hold.

$$\begin{array}{ll}
\alpha(\mathcal{B}[\![\mathcal{E}, \mathbf{G}\ in\ \mathsf{P}]\!]) = & [\text{by definition of } \mathcal{B}[\![\cdot]\!]] \\
\alpha(\bigcup\{\mathcal{E}'|_{var(\mathbf{G})} \mid \mathcal{E}, \mathbf{G} \xrightarrow[\mathsf{P}]{\mathrm{eq}}{}^* \mathcal{E}', \square\}) = & [\text{by (4.3.1)}] \\
\alpha(\bigcup\{\gamma\alpha(\mathcal{E}'|_{var(\mathbf{G})}) \mid \mathcal{E}, \mathbf{G} \xrightarrow[\mathsf{P}]{\mathrm{eq}}{}^* \mathcal{E}', \square\}) = & [\text{by (1)}] \\
\alpha(\bigcup\{\gamma((\alpha(\mathcal{E}'))\tilde{|}_{var(\mathbf{G})}) \mid \mathcal{E}, \mathbf{G} \xrightarrow[\mathsf{P}]{\mathrm{eq}}{}^* \mathcal{E}', \square\}) = & [\text{by the previous observation}] \\
\alpha(\bigcup\{\gamma(\mathsf{D}\tilde{|}_{var(\mathbf{G})}) \mid \alpha(\mathcal{E}), \mathbf{G} \xrightarrow[\mathsf{P}]{\alpha-\mathrm{eq}}{}^* \mathsf{D}, \square\}) = & [\text{by definition of } \widetilde{\bigcup}]
\end{array}$$

$$\widetilde{\bigcup}\{\tilde{D}|_{var(\mathbf{G})} \mid \alpha(\mathcal{E}), \mathbf{G} \xrightarrow[P]{\alpha-eq}{}^{*} D, \square\} = \qquad [\text{by definition of } \mathcal{B}_\alpha[\![\cdot]\!]]$$
$$\mathcal{B}_\alpha[\![\alpha(\mathcal{E}), \mathbf{G}\ in\ P]\!]$$

and therefore

$$\begin{aligned}
&\alpha(\mathcal{O}[\![P]\!]) = && [\text{by definition of } \mathcal{O}[\![\cdot]\!]]\\
&\alpha((\lambda p(\mathbf{x}).\ \mathcal{B}[\![p(\mathbf{x})\ in\ P]\!])\,/_{\equiv_{\mathbb{C}}}) = && [\text{by definition of } \equiv_{\mathbb{A}}]\\
&(\alpha(\lambda p(\mathbf{x}).\ \mathcal{B}[\![p(\mathbf{x})\ in\ P]\!]))\,/_{\equiv_{\mathbb{A}}} = && [\text{by (5) and (3)}]\\
&(\lambda p(\mathbf{x}).\ \alpha(\mathcal{B}[\![p(\mathbf{x})\ in\ P]\!]))\,/_{\equiv_{\mathbb{A}}} = && [\text{by the previous result}]\\
&(\lambda p(\mathbf{x}).\ \mathcal{B}_\alpha[\![p(\mathbf{x})\ in\ P]\!])\,/_{\equiv_{\mathbb{A}}} = && [\text{by definition of } \mathcal{O}_\alpha[\![\cdot]\!]]\\
&\mathcal{O}_\alpha[\![P]\!].
\end{aligned}$$

Point 2 Let $\mathbf{z}$ and $\mathbf{y}_i$ be the variables $var(A_1, \ldots, A_n)$ and $var(A_i)$ for $i \in [1, n]$, respectively. First of all note that since D is finite and relevant w.r.t. $\mathbf{z}$ (by α additivity) there must be a finite and relevant (w.r.t. $\mathbf{z}$) element $\mathcal{E} \in \mathbb{E}$ such that $\alpha(\mathcal{E}) = D$. Hence $\mathcal{E}, A_1, \ldots, A_n$ is an equational goal and then the following facts hold.

$$\begin{aligned}
&\mathcal{B}_\alpha[\![D, A_1, \ldots, A_n\ in\ P]\!] =\\
&\qquad [\text{by Point 1}]\\
&\alpha(\mathcal{B}[\![\mathcal{E}, A_1, \ldots, A_n\ in\ P]\!]) =\\
&\qquad [\text{by Point 1 of Theorem 4.1.2}]\\
&\alpha((\mathcal{E} \otimes^{\mathbf{z}}_{\mathbf{y}_1} \mathcal{O}[\![P]\!](A_1) \otimes^{\mathbf{z}}_{\mathbf{y}_2} \cdots \otimes^{\mathbf{z}}_{\mathbf{y}_n} \mathcal{O}[\![P]\!](A_n))|_{\mathbf{z}}) =\\
&\qquad [\text{by (1) and (2)}]\\
&(D \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_1} \alpha(\mathcal{O}[\![P]\!](A_1)) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_2} \cdots \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_n} \alpha(\mathcal{O}[\![P]\!](A_n)))\tilde{|}_{\mathbf{z}} =\\
&\qquad [\text{by (7)}]\\
&(D \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_1} (\alpha(\mathcal{O}[\![P]\!])(A_1)) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_2} \cdots \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_n} (\alpha(\mathcal{O}[\![P]\!])(A_n)))\tilde{|}_{\mathbf{z}} =\\
&\qquad [\text{by Point 1}]\\
&(D \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_1} \mathcal{O}_\alpha[\![P]\!](A_1) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_2} \cdots \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_n} \mathcal{O}_\alpha[\![P]\!](A_n))\tilde{|}_{\mathbf{z}}.
\end{aligned}$$

Point 3 The following facts hold.

$$\begin{aligned}
&\alpha(\mathcal{P}[\![P]\!](\mathcal{I}))\\
&= [\text{by definition of } \mathcal{P}[\![\cdot]\!]]\\
&\alpha(\lambda p(\mathbf{x}).\ \bigcup\{\mathcal{E} \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed clause of } P,\\
&\quad \mathbf{z} = var(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = var(A_i),\\
&\quad \mathcal{E} = (\{E\} \otimes^{\mathbf{z}}_{\mathbf{y}_1} \mathcal{I}(A_1) \otimes^{\mathbf{z}}_{\mathbf{y}_2} \cdots \otimes^{\mathbf{z}}_{\mathbf{y}_n} \mathcal{I}(A_n))|_{\mathbf{x}}\})\\
&= [\text{by (4)}]
\end{aligned}$$

$$\begin{aligned}
&\lambda p(\mathbf{x}).\, \alpha(\bigcup\{\mathcal{E} \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed clause of } P,\\
&\quad z = var(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = var(A_i),\\
&\quad \mathcal{E} = (\{E\} \otimes^z_{\mathbf{y}_1} \mathcal{I}(A_1) \otimes^z_{\mathbf{y}_2} \cdots \otimes^z_{\mathbf{y}_n} \mathcal{I}(A_n))|_{\mathbf{x}}\})\\
&= [\text{by } (4.3.1)]\\
&\lambda p(\mathbf{x}).\, \alpha(\bigcup\{\gamma(D) \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed clause of } P,\\
&\quad z = var(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = var(A_i),\\
&\quad D = \alpha((\{E\} \otimes^z_{\mathbf{y}_1} \mathcal{I}(A_1) \otimes^z_{\mathbf{y}_2} \cdots \otimes^z_{\mathbf{y}_n} \mathcal{I}(A_n))|_{\mathbf{x}})\})\\
&= [\text{by } (1) \text{ and } (2)]\\
&\lambda p(\mathbf{x}).\, \alpha(\bigcup\{\gamma(D) \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed clause of } P,\\
&\quad z = var(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = var(A_i),\\
&\quad D = (\alpha(\{E\}) \widetilde{\otimes}^z_{\mathbf{y}_1} \alpha(\mathcal{I}(A_1)) \widetilde{\otimes}^z_{\mathbf{y}_2} \cdots \widetilde{\otimes}^z_{\mathbf{y}_n} \alpha(\mathcal{I}(A_n)))\tilde{|}_{\mathbf{x}}\})\\
&= [\text{by } (7)]\\
&\lambda p(\mathbf{x}).\, \alpha(\bigcup\{\gamma(D) \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed clause of } P,\\
&\quad z = var(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = var(A_i),\\
&\quad D = (\alpha(\{E\}) \widetilde{\otimes}^z_{\mathbf{y}_1} (\alpha(\mathcal{I})(A_1)) \widetilde{\otimes}^z_{\mathbf{y}_2} \cdots \widetilde{\otimes}^z_{\mathbf{y}_n}\\
&\quad (\alpha(\mathcal{I})(A_n)))\tilde{|}_{\mathbf{x}}\})\\
&= [\text{by definition of } \widetilde{\bigcup}]\\
&\lambda p(\mathbf{x}).\, \widetilde{\bigcup}\{D \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed clause of } P,\\
&\quad z = var(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = var(A_i),\\
&\quad D = (\alpha(\{E\}) \widetilde{\otimes}^z_{\mathbf{y}_1} (\alpha(\mathcal{I})(A_1)) \widetilde{\otimes}^z_{\mathbf{y}_2} \cdots \widetilde{\otimes}^z_{\mathbf{y}_n} (\alpha(\mathcal{I})(A_n)))\tilde{|}_{\mathbf{x}}\}\\
&= [\text{by definition of } \mathcal{P}[\![\cdot]\!]]\\
&\mathcal{P}_\alpha[\![P]\!](\alpha(\mathcal{I})).
\end{aligned}$$

Point 4 Let $\{\mathcal{I}_{\alpha i}\}_{i \in I} \subseteq \mathbb{I}_{\mathbb{A}}$ be a chain. We prove that

$$\widetilde{\bigsqcup}\{\mathcal{P}_\alpha[\![P]\!](\mathcal{I}_{\alpha i})\}_{i \in I} = \mathcal{P}_\alpha[\![P]\!](\widetilde{\bigsqcup}\{\mathcal{I}_{\alpha i}\}_{i \in I}).$$

The following hold.

$$\begin{aligned}
&\widetilde{\bigsqcup}\{\mathcal{P}_\alpha[\![P]\!](\mathcal{I}_{\alpha i})\}_{i \in I} = && [\text{by definition of } \widetilde{\sqcup}]\\
&\alpha(\sqcup\{\gamma(\mathcal{P}_\alpha[\![P]\!](\mathcal{I}_{\alpha i}))\}_{i \in I}) = && [\text{by Point 3 and since } \alpha\gamma = Id]\\
&\alpha(\sqcup\{\gamma\alpha(\mathcal{P}[\![P]\!](\gamma(\mathcal{I}_{\alpha i})))\}_{i \in I}) = && [\text{by } (8)]\\
&\alpha(\sqcup\{\mathcal{P}[\![P]\!](\gamma(\mathcal{I}_{\alpha i}))\}_{i \in I}) = && [\text{since } \mathcal{P}[\![P]\!] \text{ is continuous}]\\
&\alpha(\mathcal{P}[\![P]\!](\sqcup\{\gamma(\mathcal{I}_{\alpha i})\}_{i \in I})) = && [\text{by Point 3}]\\
&\mathcal{P}_\alpha[\![P]\!](\alpha(\sqcup\{\gamma(\mathcal{I}_{\alpha i})\}_{i \in I})) = && [\text{by definition of } \widetilde{\sqcup}]
\end{aligned}$$

$\mathcal{P}_\alpha[\![P]\!](\widetilde{\bigsqcup}\{\mathcal{I}_{\alpha i}\}_{i\in I})$.

Point 5 First of all observe that by Point 4 and a straightforward inductive argument, $\forall n \geq 0.\, \alpha(\mathcal{P}[\![P]\!]\uparrow_n) = \mathcal{P}_\alpha[\![P]\!]\uparrow_n$. Therefore, the following equalities hold.

$$
\begin{array}{ll}
\alpha(\mathcal{F}[\![P]\!]) = & [\text{by definition of } \mathcal{F}[\![\cdot]\!]] \\
\alpha(\mathcal{P}[\![P]\!]\uparrow_\omega) = & [\mathcal{P}[\![P]\!] \text{ is continuous}] \\
\alpha(\sqcup\{\mathcal{P}[\![P]\!]\uparrow_n\}_{n\geq 0}) = & [\text{by } (8)] \\
\alpha(\sqcup\{\gamma\alpha(\mathcal{P}[\![P]\!]\uparrow_n)\}_{n\geq 0}) = & [\text{by definition of } \widetilde{\sqcup}] \\
\widetilde{\bigsqcup}\{\alpha(\mathcal{P}[\![P]\!]\uparrow_n)\}_{n\geq 0} = & [\text{by the previous observation}] \\
\widetilde{\bigsqcup}\{\mathcal{P}_\alpha[\![P]\!]\uparrow_n\}_{n\geq 0} = & [\mathcal{P}_\alpha[\![P]\!] \text{ is continuous}] \\
\mathcal{P}_\alpha[\![P]\!]\uparrow_\omega = & [\text{by definition of } \mathcal{F}_\alpha[\![\cdot]\!]] \\
\mathcal{F}_\alpha[\![P]\!]\,. &
\end{array}
$$

Point 6 Straightforward by Point 3 of Theorem 4.1.2 and by Points 1 and 5. ■

Proof of Theorem 4.4.2. First of all note that a more general property of (4.6.5) holds. Namely, for any $\mathbf{x}, \mathbf{y}, \mathbf{z}$ such that $\mathbf{x} \cup \mathbf{y} \subseteq \mathbf{z}$,

$$(\mathcal{E}_1 \otimes^{\mathbf{z}}_{\mathbf{y}} \mathcal{E}_2)|_{\mathbf{x}} = (\mathcal{E}_1|_{\mathbf{z}} \otimes^{\mathbf{z}}_{\mathbf{y}} \mathcal{E}_2)|_{\mathbf{x}}. \tag{1}$$

We prove Point 4.

$$
\begin{array}{l}
\alpha(\mathcal{P}[\![P]\!](\gamma(\mathcal{I}_\alpha))) \\
= [\text{by definition of } \alpha] \\
\lambda p(\mathbf{x}).\, (\alpha(\mathcal{P}[\![P]\!](\gamma(\mathcal{I}_\alpha))))\tilde{|}_{\mathbf{x}} \\
= [\text{by definition of } \mathcal{P}[\![\cdot]\!]] \\
\lambda p(\mathbf{x}).\, (\alpha(\bigcup\{\mathcal{E} \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed} \\
\quad \text{clause of } P,\ \mathbf{z} = \mathit{var}(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = \mathit{var}(A_i), \\
\quad \mathcal{E} = (\{E\} \otimes^{\mathbf{z}}_{\mathbf{y}_1} \gamma(\mathcal{I}_\alpha)(A_1) \otimes^{\mathbf{z}}_{\mathbf{y}_2} \cdots \otimes^{\mathbf{z}}_{\mathbf{y}_n} \gamma(\mathcal{I}_\alpha)(A_n))|_{\mathbf{x}}\}))\tilde{|}_{\mathbf{x}} \\
= [\text{by definition of } \gamma] \\
\lambda p(\mathbf{x}).\, (\alpha(\bigcup\{\mathcal{E} \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed} \\
\quad \text{clause of } P,\ \mathbf{z} = \mathit{var}(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = \mathit{var}(A_i), \\
\quad \mathcal{E} = (\{E\} \otimes^{\mathbf{z}}_{\mathbf{y}_1} \gamma(\mathcal{I}_\alpha(A_1))|_{\mathbf{y}_1} \otimes^{\mathbf{z}}_{\mathbf{y}_2} \cdots \otimes^{\mathbf{z}}_{\mathbf{y}_n} \gamma(\mathcal{I}_\alpha(A_n))|_{\mathbf{y}_n})|_{\mathbf{x}}\}))\tilde{|}_{\mathbf{x}} \\
= [\text{since } \mathbf{y}_i \cup \mathbf{x} \subseteq \mathbf{z}, \text{ by } (1) \text{ and } (4.6.5)]
\end{array}
$$

$$
\begin{aligned}
&\lambda p(\mathbf{x}).\,(\alpha(\bigcup\{\mathcal{E} \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed}\\
&\quad \text{clause of } P,\ \mathbf{z} = \mathit{var}(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = \mathit{var}(A_i),\\
&\quad \mathcal{E} = (\{E\} \otimes^{\mathbf{z}}_{\mathbf{y}_1} \gamma(\mathcal{I}_\alpha(A_1)) \otimes^{\mathbf{z}}_{\mathbf{y}_2} \cdots \otimes^{\mathbf{z}}_{\mathbf{y}_n} \gamma(\mathcal{I}_\alpha(A_n)))|_{\mathbf{x}}\})\tilde{|}_{\mathbf{x}}\\
&= [\text{by (4.3.1) and by definition of } \widetilde{\cup}]\\
&\lambda p(\mathbf{x}).\ \widetilde{\bigcup}\{D \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed}\\
&\quad \text{clause of } P,\ \mathbf{z} = \mathit{var}(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = \mathit{var}(A_i),\\
&\quad D = \alpha((\{E\} \otimes^{\mathbf{z}}_{\mathbf{y}_1} \gamma(\mathcal{I}_\alpha(A_1)) \otimes^{\mathbf{z}}_{\mathbf{y}_2} \cdots \otimes^{\mathbf{z}}_{\mathbf{y}_n} \gamma(\mathcal{I}_\alpha(A_n)))|_{\mathbf{x}})\}\tilde{|}_{\mathbf{x}}\\
&= [\text{since } \tilde{|}_{\mathbf{x}} \text{ distributes over sums}]\\
&\lambda p(\mathbf{x}).\ \widetilde{\bigcup}\{D \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed}\\
&\quad \text{clause of } P,\ \mathbf{z} = \mathit{var}(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = \mathit{var}(A_i),\\
&\quad D = (\alpha((\{E\} \otimes^{\mathbf{z}}_{\mathbf{y}_1} \gamma(\mathcal{I}_\alpha(A_1)) \otimes^{\mathbf{z}}_{\mathbf{y}_2} \cdots \otimes^{\mathbf{z}}_{\mathbf{y}_n} \gamma(\mathcal{I}_\alpha(A_n)))|_{\mathbf{x}}))\tilde{|}_{\mathbf{x}}\}\\
&= [\text{by Point 7 of Definition 4.4.1}]\\
&\lambda p(\mathbf{x}).\ \widetilde{\bigcup}\{D \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed}\\
&\quad \text{clause of } P,\ \mathbf{z} = \mathit{var}(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = \mathit{var}(A_i),\\
&\quad D = (\alpha(\{E\} \otimes^{\mathbf{z}}_{\mathbf{y}_1} \gamma(\mathcal{I}_\alpha(A_1)) \otimes^{\mathbf{z}}_{\mathbf{y}_2} \cdots \otimes^{\mathbf{z}}_{\mathbf{y}_n} \gamma(\mathcal{I}_\alpha(A_n))))\tilde{|}_{\mathbf{x}}\}\\
&= [\text{using repeatedly Point 1 of Definition 4.4.1 and } \widetilde{\otimes} \text{ definition}]\\
&\lambda p(\mathbf{x}).\ \widetilde{\bigcup}\{D \mid c = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed}\\
&\quad \text{clause of } P,\ \mathbf{z} = \mathit{var}(c), \text{ for } i \in [1, n],\ \mathbf{y}_i = \mathit{var}(A_i),\\
&\quad D = (\alpha(\{E\}) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_1} \mathcal{I}_\alpha(A_1) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_2} \cdots \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_n} \mathcal{I}_\alpha(A_n))\tilde{|}_{\mathbf{x}}\}\\
&= [\text{by } \mathcal{P}_\alpha[\![\cdot]\!] \text{ definition}]\\
&\mathcal{P}_\alpha[\![P]\!](\mathcal{I}_\alpha).
\end{aligned}
$$

Points 1 and 5 are a straightforward consequence of correctness (extensivity) of the insertion $\alpha : \mathbb{E} \to \mathbb{D}$.

For the other points, first of all note that Point 1 of Definition 4.4.1 implies $\widetilde{\otimes}$, $\widetilde{\otimes}^{\mathbf{y}}_{\mathbf{x}}$ additivity, $\widetilde{\otimes}$ associativity and the abstract counterparts of equations (4.6.2), (4.6.3) and (4.6.4) of the proof of Theorem 4.1.2. Point 2 of Definition 4.4.1 ensures abstract additivity of $\tilde{|}$.

Furthermore note that, by Point 1 of Definition 4.4.1 (taking $\mathbf{x} = V$) and by definition of the abstract operation,

$$(\alpha(\{\emptyset\}) \widetilde{\otimes} D)\tilde{|}_{\mathbf{x}} = D\tilde{|}_{\mathbf{x}}. \tag{2}$$

Points 3, 4, 5 and 6 (of Definition 4.4.1) and (2) are the abstract versions of equations (4.6.1), (4.6.5), (4.6.6), (4.6.7) and (4.6.3) respectively. Moreover, α and γ are

congruences w.r.t. renaming. Hence the proof of this theorem is the same as the proof of Theorem 4.1.2. ■

Chapter 5

Abstract Diagnosis

5.1 Introduction

5.1.1 Declarative Debugging, Program Verification and Abstract Diagnosis

Declarative debugging [91, 74, 48] is a technique which, given a program P and a specification $\mathcal{I}$ of the *intended* declarative semantics of P, allows one to determine program bugs, when the actual semantics $\mathcal{F}[\![\mathsf{P}]\!]$ and the specification $\mathcal{I}$ are different. Declarative debugging algorithms are based on a theory which requires $\mathcal{I}$ to be specified extensionally. However, since $\mathcal{I}$ is in general infinite, practical debugging algorithms are driven by symptoms (atoms on which $\mathcal{F}[\![\mathsf{P}]\!]$ and $\mathcal{I}$ do not agree), which are determined by using testing techniques. Oracles are used to model the acquisition (from the user) of the subset of $\mathcal{I}$ which is relevant to a symptom. Given a symptom, the algorithms query the oracles to locate the actual sources of errors.

One stronger alternative to symptom-directed declarative debugging can be obtained by extending the underlying theory to the case where the specification $\mathcal{I}$ is a finite representation of the intended behavior. This is essentially the approach of *program verification* [40, 12, 5, 4], where the specification is a finite (intensional) representation of a program property. The property is any abstraction of the semantics, including the semantics itself. The goal of program verification is proving that the program is partially correct, i.e., that it satisfies the specification. It is worth noting that in program verification a specification is usually a pair of pre- and post-conditions. The property specified by the post-condition has to be satisfied only by those goals which satisfy the pre-condition. In addition the partial correctness criteria might require all the procedure calls (call patterns) to satisfy their pre-conditions.

The aim of *abstract diagnosis* is to extend declarative debugging to the case where

specifications are finite and define a program property rather than its semantics. In order to be consistent with traditional declarative debugging, specifications are assumed to consist of post-conditions only. However the results can be generalized to the case of pre- and post-conditions (see Section 5.7). Finite specifications lead to the systematic derivation of the diagnosis algorithms from the underlying theory with *no need for symptom detection*. Moreover, the theoretical results on partial correctness, completeness and bug derivation are valid for the diagnosis algorithms too. The approach of abstract diagnosis is strongly related to the idea of using assertions as finite specifications of an approximation of the intended declarative semantics in [41] and to the concept of an abstract oracle, introduced in [73] to specify a superset of the intended program behavior, in the case of concurrent logic programs.

5.1.2 Program Properties and Abstract Interpretation

Program properties are formulas in a logical theory. They can be viewed as abstractions of a suitable semantics. Their relation to the semantics can be formalized within abstract interpretation theory [32, 34]. However, abstract interpretation suggests another way of looking at program properties, where the logical theory is replaced by a finite (or Noetherian) model (the abstract domain). The relevant feature of abstract interpretation is that, once the property has been modeled by an abstract domain, we have a methodology to systematically derive an abstract semantics, which in turn allows us to effectively compute a (correct) approximation of the property. By using this approach, most of the theorem-proving, in the logical theory involved in program verification, boils down to computing on the abstract domain. This is obtained in general at the expense of precision.

In program analysis, abstract interpretation theory is often used to establish the correctness of specific analysis algorithms and abstract domains. We are more concerned instead in its application to the systematic derivation of the (optimal) abstract semantics from the abstract domain. Recent results on domain refinement operators (see, for example, [50, 62, 89]) show that (optimal) abstract domains can systematically be derived from the property to be proved.

5.1.3 The Semantic Framework

Program properties we are interested in are operational properties and not necessarily declarative properties. The aim of the resulting method is therefore closer to the goal of rational debugging [85] than to the goal of declarative debugging. This also means that we cannot base the abstraction framework on the declarative semantics,

since it is too abstract to allow us to reason about some operational properties, such as variable groundness.

Some preliminary versions of abstract diagnosis [25, 27] were based on the semantic framework of Part I. In this chapter we are only concerned with properties which are abstractions of the computed answer semantics. Furthermore we are interested in equivalent top-down and bottom-up semantics, since this will allow us to define equivalent top-down and bottom-up diagnosis algorithms. We will therefore use the applicative semantic framework of Chapter 4.

Program properties are *observables*, i.e., Galois insertions between the concrete domain (the semantic domain of the collecting semantics) and the abstract domain chosen to model the property. The abstract semantics (abstract transition system and abstract denotational semantics) are systematically derived from the collecting semantics and the observable. We consider two classes of observables, *complete* and *approximate*. For every complete or approximate observable, the abstract operational semantics and the abstract denotational semantics are equivalent. This will allow us to define equivalent top-down and bottom-up diagnosis algorithms. The above equivalence property requires the observable to be *condensing*. Condensing is a compositionality property which tells us that the abstract semantics of a procedure call can be derived (without losing precision) from the abstract semantics of the procedure declaration. This property is needed in abstract diagnosis where the specification is a post-condition describing a (goal-independent) property of a set of procedure declarations. It is worth noting that the observables corresponding to the declarative semantics are condensing and that the declarative semantics do indeed characterize procedure declarations. Note also that several observables used in program analysis (for mode, type and groundness analysis) are also condensing and that a non-condensing observable can systematically be transformed into a (more concrete) condensing observable, by using domain refinement operators (see, for example, how the condensing domain $\mathcal{POS}$ for groundness analysis can be derived [89] from the non-condensing domain $\mathcal{DEF}$).

As expected from their definitions in Chapter 4, the difference between complete and approximate observables is related to precision. Namely, the abstract semantics coincides with the abstraction of the collecting semantics, in the case of complete observables, while it is just a correct approximation, in the case of approximate observables. On the other side, approximate observables correspond to Noetherian domains. Hence their abstract semantics is finite, while (in general) it is infinite for complete observables. The class of complete observables includes the observables ground instances of computed answers and correct answers which allow us to reconstruct the declarative semantics used in declarative debugging, i.e., the least Herbrand model used in [91] and the least term model (atomic logical consequences

or c-semantics) used in [48]. On the other hand, the class of approximate observables includes *depth*(k) [88] and several domains proposed for type, mode and groundness analysis (we will just consider the domain $\mathcal{POS}$ [8] for groundness analysis).

5.1.4 Abstract Diagnosis

In Section 5.2 we give the basic definitions of abstract diagnosis, which are a straightforward adaptation of those given for declarative debugging. A preliminary version (without proofs) of abstract diagnosis can be found in [25, 27, 24]. *Partial correctness* and *completeness* of P w.r.t. the observable property α are defined by comparing the abstract specification of the *intended* behavior (of P w.r.t. α) $\mathcal{I}_\alpha$ and the abstraction $\alpha(\mathcal{F}[\![\mathsf{P}]\!])$ of the concrete semantics $\mathcal{F}[\![\mathsf{P}]\!]$. It is worth noting that $\alpha(\mathcal{F}[\![\mathsf{P}]\!])$ is in general more precise than the abstract semantics $\mathcal{F}_\alpha[\![\mathsf{P}]\!]$, in the case of approximate observables.

The *diagnosis* is based on the detection of *incorrect clauses* and *uncovered elements*, which have both a bottom-up definition (in terms of one application of the "abstract immediate consequence operator" to the abstract specification, see Section 5.2.1) and a top-down definition (in terms of "oracle simulation", see Section 5.2.2). It is worth noting that both the definitions use the (possibly approximate) computation on the abstract domain, and that no fixpoint computation is required, since the abstract semantics does not need to be computed.

An implementation of the diagnosis algorithms (parametric w.r.t. the observable) by means of PROLOG meta-programs is described in Section 5.2.3.

In Section 5.3 we give the diagnosis theorems for complete observables, which provide rather strong results. Namely,

- absence of incorrect clauses implies partial correctness,
- absence of uncovered elements implies completeness, for a large class of programs (acceptable programs),
- incorrect clauses and uncovered elements always correspond to a bug in the program.

The results for the complete observable generalize the ones stated in the literature for declarative debugging and allow us to reconstruct the theory of declarative debugging as an instance of abstract diagnosis. In addition we have some new stronger results on the diagnosis of completeness. However, since abstract specifications are often infinite in the case of complete observables, these results have a purely theoretical interest and can be viewed as a foundation for the effective diagnosis methods considered in the following sections.

The first effective diagnosis method (*partial diagnosis*) is described in Section 5.4. Partial diagnosis (originally introduced in [26]) can be applied to make the diagnosis

effective in the case of complete observables. A specification consists of a finite set of elements which *are* in the intended behavior and a finite set of elements which *are not* in the intended behavior. The diagnosis is based on the detection of p-incorrect clauses and p-uncovered elements. The results we obtain are of course weaker. Namely,

- p-incorrect clauses always correspond to bugs in the program,
- absence of p-uncovered elements implies completeness (w.r.t. the positive specification) for a large class of programs,
- a p-uncovered element is a *warning* about a possible incompleteness bug.

We show that partial diagnosis can be viewed as a theoretical foundation of the symptom-directed oracle-based debugging algorithms used in declarative debugging.

The second effective diagnosis method can be applied if the property can be modeled by approximate observables. For example, one can choose to approximate the complete observable computed answers by the (approximate) observable $depth(\mathtt{k})$ answers, which leads to finite abstract specifications. Using abstract diagnosis w.r.t. approximate observables one can effectively prove properties related to modes, types and groundness dependencies. The results for approximate observables are given in Section 5.5. Again, the results are weaker than those for complete observables (because of approximation). Namely,

- absence of incorrect clauses implies partial correctness,
- an uncovered element *always* corresponds to a bug in the program,
- *all* the incorrectness bugs are captured by incorrect clauses
- an incorrect clause is a *warning* about a possible incorrectness bug.

It is worth noting that there exists a duality between the results for partial diagnosis and those for approximate observables. This is due to the fact that a partial specification is a subset of the full specification, while an abstract specification corresponding to an approximate observable represents (through the concretization function) a superset of the concrete full specification.

In Section 5.6 we consider the problem of modular diagnosis and we formally prove that the diagnosis method does not need to be extended to perform the diagnosis in a modular way. This is due to the fact that both the top-down and the bottom-up diagnosis algorithms are essentially based on the application of the "abstract immediate consequence operator" which is intrinsically compositional. This property shows that we can verify and debug incomplete programs, once we have the specifications for the missing program components.

Finally in Section 5.7 we compare the results of abstract diagnosis w.r.t. approximate observables to those of program verification (with post-conditions only).

It turns out that the results are similar for partial correctness and completeness. Abstract diagnosis is more useful for debugging purposes, since it provides useful information on program bugs.

The results of this chapter are from [23, 29, 24, 26, 28, 27].

5.2 Abstract Diagnosis

In the following, the observable α will always be assumed to be at least approximate, since we know that for these observables the actual and the intended behaviors for all the goals of a program are uniquely determined by the behaviors for pure atomic goals. The following Definition 5.2.1 extends to abstract diagnosis the definitions given in [91, 48, 74] for declarative debugging. In the following $\mathcal{I}_\alpha$ is the specification of the intended behavior of a program for pure atoms w.r.t. the observable α.

Definition 5.2.1 *Let* P *be a program and* α *be an observable.*

1. P *is* partially correct *w.r.t.* $\mathcal{I}_\alpha$ *if* $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) \leq \mathcal{I}_\alpha$.
2. P *is* complete *w.r.t.* $\mathcal{I}_\alpha$ *if* $\mathcal{I}_\alpha \leq \alpha(\mathcal{F}[\![\mathsf{P}]\!])$.
3. P *is* totally correct *w.r.t.* $\mathcal{I}_\alpha$, *if* $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{I}_\alpha$.

It is worth noting that the above definition is given in terms of the abstraction of the concrete semantics $\alpha(\mathcal{F}[\![\mathsf{P}]\!])$ and not in terms of the (possibly less precise) abstract semantics $\mathcal{F}_\alpha[\![\mathsf{P}]\!]$. This means that $\mathcal{I}_\alpha$ is the abstraction of the intended concrete semantics of P. In other words, the specifier can only reason in terms of the properties of the expected concrete semantics and cannot be concerned with (approximate) abstract computations. Note also that our notion of total correctness does not concern termination. We cannot address termination issues here, since the concrete semantics we use is too abstract.

If P is not totally correct, we are left with the problem of determining the errors, which are related to the *symptoms*. Symptoms are $\mathbb{A}$-*elements*, according to the following definition.

Definition 5.2.2 *An* $\mathbb{A}$-element σ *is an* $\mathbb{A}$*-interpretation defined for a pure atom* A *only (and is otherwise undefined). Furthermore, if the images of the elements of* $\mathbb{A}$ *(which are functions) are structured as a set, then* $\sigma(\mathsf{A})$ *must also be a singleton.*

By abuse of notation, in the following $\mathsf{A} \mapsto \{\mathsf{E}\}$ will be denoted simply by $\mathsf{A} \mapsto \mathsf{E}$.

Definition 5.2.3 *Let* P *be a program and* α *be an observable. Then*

1. *An* incorrectness symptom *is an* $\mathbb{A}$*-element* σ *such that* $\sigma \leq \alpha(\mathcal{F}[\![P]\!])$ *and* $\sigma \not\leq \mathcal{I}_\alpha$.

2. *An* incompleteness symptom *is an* $\mathbb{A}$*-element* σ *such that* $\sigma \leq \mathcal{I}_\alpha$ *and* $\sigma \not\leq \alpha(\mathcal{F}[\![P]\!])$.

Note that a totally correct program has no incorrectness and no incompleteness symptoms. Our incompleteness symptoms are related to the insufficiency symptoms in [48], which are defined by taking *gfp* $\mathcal{P}[\![P]\!]$ instead of $\mathcal{F}[\![P]\!] = \mathit{lfp}\, \mathcal{P}[\![P]\!]$ as program semantics. The two definitions, even if different, turn out to be the same for the class of acceptable programs (see Section 5.3.1). Ferrand's choice is motivated by the fact that *gfp* $\mathcal{P}[\![P]\!]$ is related to finite failures. The approach of using two different semantics for reasoning about incorrectness and incompleteness has been pursued in [49], leading to an elegant uniform (yet non-effective) characterization of correctness and completeness.

It is straightforward to realize that an $\mathbb{A}$-element may sometimes be an incorrectness or incompleteness symptom, just because of another symptom.

Example 5.2.4

Consider the program P of Figure 5.1 w.r.t. the computed answer observable and the specification

$$\mathcal{I} := \begin{cases} p(x) \mapsto \emptyset \\ q(x) \mapsto \emptyset. \end{cases}$$

The denotation of P is

$$\mathcal{F}[\![P]\!] = \begin{cases} p(x) \mapsto \{x = a\} \\ q(x) \mapsto \{x = a\} \end{cases}$$

Hence $\sigma_1 := p(x) \mapsto \{x = a\}$ and $\sigma_2 := q(x) \mapsto \{x = a\}$ are both incorrectness symptoms but σ_2 is just a consequence of σ_1.

Consider now the specification $\mathcal{I}' := q(x) \mapsto \{x = a\}$. There exists only an incorrectness symptom, i.e., $\sigma := p(x) \mapsto \{x = a\}$. If we fix this bug (by removing the second clause), we get an incompleteness symptom, since for the modified program Q, $\mathcal{F}[\![Q]\!](q(x)) = \emptyset$.

The *diagnosis* determines the "basic" symptoms and, in the case of incorrectness, the relevant clause in the program. This is captured by the definitions of *incorrect clause* and *uncovered* $\mathbb{A}$*-element*, which are related to incorrectness and incompleteness symptoms, respectively. As we will show in the next two subsections, incorrect clauses and uncovered $\mathbb{A}$-elements can equivalently be characterized in a bottom-up and top-down way.

```
c1:  q(X) :- p(X).
c2:  p(a).
```

Figure 5.1: The program of Examples 5.2.4 and 5.2.7

5.2.1 Bottom-up Diagnosis

The bottom-up diagnosis is based on the application of the *abstract immediate consequence operator* $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$.

Definition 5.2.5 *Let* P *be a program. If there exists an* $\mathbb{A}$*-element* σ *such that* $\sigma \not\leq \mathcal{I}_\alpha$ *and* $\sigma \leq \mathcal{P}_\alpha[\![\{\mathsf{c}\}]\!](\mathcal{I}_\alpha)$[1]*, then the clause* $\mathsf{c} \in \mathsf{P}$ *is* incorrect *on* σ*.*

Informally, c is incorrect on σ if it derives a wrong $\mathbb{A}$-element from the intended semantics.

Definition 5.2.6 *Let* P *be a program. An* $\mathbb{A}$*-element* σ *is* uncovered *if* $\sigma \leq \mathcal{I}_\alpha$ *and* $\sigma \not\leq \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha)$*.*

Informally, σ is uncovered if there are no clauses deriving it from the intended semantics.

Example 5.2.7

Consider (again) the program of Figure 5.1 w.r.t. the observable computed answers and the specification $\mathcal{I}$ of Example 5.2.4. It shows that $\sigma_1 := \mathsf{p}(\mathsf{x}) \mapsto \{\mathsf{x} = \mathsf{a}\}$ and $\sigma_2 := \mathsf{q}(\mathsf{x}) \mapsto \{\mathsf{x} = \mathsf{a}\}$ are both incorrectness symptoms (even if σ_2 is just a consequence of σ_1). By applying Definition 5.2.5 we obtain

$$\mathcal{P}[\![\{\mathsf{c1}\}]\!](\mathcal{I}) = \mathsf{q}(\mathsf{x}) \mapsto \emptyset$$
$$\mathcal{P}[\![\{\mathsf{c2}\}]\!](\mathcal{I}) = \mathsf{p}(\mathsf{x}) \mapsto \{\mathsf{x} = \mathsf{a}\}.$$

Hence we detect one bug only, i.e., that the clause $\mathsf{c2}$ is incorrect on σ_1.

Consider now the specification $\mathcal{I}'$ of Example 5.2.4. It shows one incorrectness symptom only, i.e., $\sigma := \mathsf{p}(\mathsf{x}) \mapsto \{\mathsf{x} = \mathsf{a}\}$. By applying Definitions 5.2.5 and 5.2.6 we find out that the clause $\mathsf{c2}$ is incorrect on σ and, in addition, that $\sigma' := \mathsf{q}(\mathsf{x}) \mapsto \{\mathsf{x} = \mathsf{a}\}$ is uncovered. This is exactly what we would obtain once we fix the incorrectness bug.

Note that (since $\mathcal{I}' \leq \mathcal{F}[\![\mathsf{P}]\!]$) the program is complete (i.e., there are no incompleteness symptoms) even if there is an uncovered $\mathbb{A}$-element.

[1] Note that $\mathcal{P}_\alpha[\![\{\mathsf{c}\}]\!]$ is the operator associated to the program $\{\mathsf{c}\}$, consisting of the clause c only.

It is worth noting that checking the conditions of Definitions 5.2.5 and 5.2.6 requires one application of $\mathcal{P}_\alpha[\![P]\!]$ to $\mathcal{I}_\alpha$, while the detection of symptoms (Definition 5.2.3) would require the construction of $\alpha(\mathcal{F}[\![P]\!])$ and therefore a fixpoint computation. The above examples suggest that bugs might be captured by incorrect clauses and uncovered $\mathbb{A}$-elements much better than by symptoms. This will be formally proved in Sections 5.3 and 5.5 for complete and approximate observables separately. However, we will now first look at an alternative characterization of incorrect clauses and uncovered $\mathbb{A}$-elements.

5.2.2 Top-down Diagnosis

The "bottom-up" diagnosis is based on Definitions 5.2.5 and 5.2.6 and requires the application of $\mathcal{P}_\alpha[\![P]\!]$ to the specification $\mathcal{I}_\alpha$. In the top-down diagnosis, $\mathcal{I}_\alpha$ can more naturally be viewed as an *abstract oracle*, i.e., it can be implemented by querying the user. Several oracles have been used in declarative debugging (see the discussion in [81]). The abstract oracle implementation of $\mathcal{I}_\alpha$ can be modeled as a function which, whenever applied to a pure atom (the query to the user), returns the abstraction of the set of all the intended computed answers.

Once we have the oracle, we can define the *abstract oracle simulation*, following [91]. The oracle simulation allows us to express in a compact way new top-down diagnosis conditions. The oracle simulation performs one step of abstract goal rewriting by using the program clauses and then gets the abstract answers for the resulting abstract goal from the oracle.

Definition 5.2.8 *Let* P *be a program. Then the* abstract oracle simulation $\mathcal{OS}_\alpha[\![\cdot]\!]$ *of* P *is*[2]

$$\begin{aligned}\mathcal{OS}_\alpha[\![P]\!] := \lambda p(\mathbf{x}).\ \widetilde{\bigcup}\big\{D \mid p(\mathbf{x}) \xrightarrow[c]{\alpha-eq} D', A_1, \ldots, A_n,\ c \textit{ is a renamed clause}\\ \textit{of } P,\ \mathbf{z} = \mathit{var}(c),\ \textit{for } i \in [1,n],\ \mathbf{y}_i = \mathit{var}(A_i),\\ D = (\alpha(\{E\}) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_1} \mathcal{I}_\alpha(A_1) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_2} \cdots \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_n} \mathcal{I}_\alpha(A_n))\tilde{\restriction}_{\mathbf{x}}\big\}.\end{aligned}$$

The following two theorems justify the top-down diagnosis.

Theorem 5.2.9 *Let* α *be either a complete or an approximate observable. A clause* $c \in P$ *is incorrect on the* $\mathbb{A}$*-element* σ *if and only if* $\sigma \leq \mathcal{OS}_\alpha[\![\{c\}]\!]$ *and* $\sigma \not\leq \mathcal{I}_\alpha$.

Proof. Let $p(\mathbf{x})$ be the (only) goal on which σ is defined and let c be a clause.

[2]Note that the $\mathbb{A}$-elements of the collections computed by $\mathcal{OS}_\alpha[\![P]\!]$ are equivalence classes w.r.t. variance, as was the case for the domain of our semantics.

Then, by Definition 5.2.8,

$$\begin{aligned}\mathcal{OS}_\alpha[\![\{c\}]\!](p(\mathbf{x})) = \widetilde{\bigcup}\{D \mid p(\mathbf{x}) \xrightarrow[c']{\alpha-eq} (\alpha(\{\emptyset\}) \widetilde{\otimes} \alpha(\{E\})), A_1, \ldots, A_n,\\ c' = p(\mathbf{x}) \leftarrow E, A_1, \ldots, A_n \text{ is a renamed version of } c,\\ \mathbf{z} = var(c'), \text{ for } i \in [1, n],\ \mathbf{y}_i = var(A_i),\\ D = ((\alpha(\{\emptyset\}) \widetilde{\otimes} \alpha(\{E\})) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_1} \mathcal{I}_\alpha(A_1) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_2} \cdots \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}_n}\\ \mathcal{I}_\alpha(A_n))\tilde{|}_{\mathbf{x}}\}.\end{aligned}$$

Note that, for any complete and approximate observable,[3] $(\alpha(\{\emptyset\}) \widetilde{\otimes} \alpha(\{E\})) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}} D' = \alpha(\{E\}) \widetilde{\otimes}^{\mathbf{z}}_{\mathbf{y}} D'$ and $(\alpha(\{\emptyset\}) \widetilde{\otimes} D')\tilde{|}_{\mathbf{x}} = D\tilde{|}_{\mathbf{x}}$. Hence, by definition of $\mathcal{P}_\alpha[\![\cdot]\!]$, $\mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}_\alpha)(p(\mathbf{x})) = \mathcal{OS}_\alpha[\![\{c\}]\!](p(\mathbf{x}))$ and therefore $\sigma \leq \mathcal{OS}_\alpha[\![\{c\}]\!]$ if and only if $\sigma \leq \mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}_\alpha)$. ■

Note that the proof of Theorem 5.2.9 is clearly based on the properties of our semantics, which relate fixpoint bottom-up computations to top-down refutations for pure atomic goals. By similar arguments we can prove the following theorem.

Theorem 5.2.10 *An $\mathbb{A}$-element σ is uncovered if and only if $\sigma \leq \mathcal{I}_\alpha$ and $\sigma \not\leq \mathcal{OS}_\alpha[\![P]\!]$.*

The top-down diagnosis definitions are particularly important, since they can naturally be implemented by meta-interpreters.

5.2.3 The Diagnosis Meta-interpreters

Our meta-interpreters[4] are shown in Figure 5.2. They are parametric w.r.t. the observable (the parameter `Obs` in the two main procedures), whose operations have to be specified in a suitable module. Apart from being generic, the meta-interpreters are very similar to those proposed for declarative debugging (see [81] for a comprehensive description). The main difference is that they do not need to start from symptoms. In fact, as expected from the definitions in Section 5.2.2, the oracle simulation just needs to be applied to finitely many pure atomic goals (generated by the call to `userdefined/1`). If the oracle returns finitely many answers to each query (i.e., if $\mathcal{I}_\alpha$ is finite), the meta-interpreters systematically derive all the incorrect clauses and uncovered $\mathbb{A}$-elements.

Let us briefly explain the base meta-interpreter code.

`incorrectClause/3`. As specified by the oracle simulation (Definition 5.2.8) we have to perform a top-down resolution step for every pure atomic goal. Hence we generate an atom and choose a clause $H \leftarrow D, B$ unifying with it

[3] The theorem does indeed hold for any observable satisfying these property.

[4] The sources of the meta-interpreters can be found in [19].

```
incorrectClause(Obs, RealElem, Clause) :-
    userdefined(Atom),
    observableClause(Obs, Atom, ClauseE, Body, Clause),
    meetTheAnswers(Obs, Body, ClauseE, RealBodyEs),
    domainProject(Obs, Atom, RealBodyEs, RealEs),
    domainSingleton(Obs, RealE, RealEs),
    not(  (observableAnswers(Obs, Atom, IntendedEs),
          domainSingleton(Obs, RealE, IntendedEs))    ),
    showAelement(Obs, Atom, RealE, RealElem).

uncoveredE(Obs, IntendedElem) :-
    userdefined(Atom),
    observableAnswers(Obs, Atom, IntendedEs),
    domainSingleton(Obs, IntendedE, IntendedEs),
    not( (  findall(  (R, ClauseE, Body),
        (observableClause(Obs,Atom,ClauseE,Body,_),R=Atom), Bodies),
      joinTheAnswers(Obs,  Atom, Bodies, RealEs),
      domainSingleton(Obs, IntendedE, RealEs)   ) ),
    showAelement(Obs, Atom, IntendedE, IntendedElem).

meetTheAnswers(Obs, true, E, E) :- !.
meetTheAnswers(Obs, (Atom,Atoms), E, E2) :- !,
    observableAnswers(Obs, Atom, AtomEs),
    domainMeet(Obs, E, AtomEs, E1),
    meetTheAnswers(Obs, Atoms, E1, E2).
meetTheAnswers(Obs, Atom, E, E1) :-
    observableAnswers(Obs, Atom, AtomEs),
    domainMeet(Obs, E, AtomEs, E1).

joinTheAnswers(Obs, _, [], Bottom) :- domainBottom(Obs, Bottom).
joinTheAnswers(Obs, Atom, [(Atom, ClauseE, Body)|Bodies], Es):-
    meetTheAnswers(Obs, Body, ClauseE, RealBodyEs),
    domainProject(Obs, Atom, RealBodyEs, SingleEs),
    joinTheAnswers(Obs, Atom, Bodies, PartialEs),
    domainJoin(Obs,SingleEs,PartialEs,Es).
```

Figure 5.2: The main module of the diagnosis programs

```
oracle(Obs, A, E) :- oracleMemory(Obs, A, E), E \== no_more_ans.
oracle(Obs, A, E) :- not( oracleMemory(Obs, A, no_more_ans) ),
    oracleAsk(Obs, A, E).
oracleAsk(Obs, A, E) :- nl,write('Any (other) answer for goal '),
    write(A),write(' wrt obs. '),write(Obs),write('?'),nl,write('>'),
    readObs(Obs,A,E),(E \== no -> asserta( (oracleMemory(Obs,A,E)) );
        assert( (oracleMemory(Obs, A, no_more_ans)) ), fail).
oracleAsk(Obs, A, E) :- not( oracleMemory(Obs, A, no_more_ans) ),
    oracleAsk(Obs, A, E).
```

Figure 5.3: The common oracle interface of the observables modules

```
userdefined(Atom),
observableClause(Obs, Atom, ClauseE, Body, Clause),
```

then we (incrementally) get from the oracle a solution for (all the atoms in) the body of the clause and we build (step by step) the meet of D and the solutions. Then we project away all uninteresting variables from the abstract resulting equation.

```
meetTheAnswers(Obs, Body, ClauseE, RealBodyEs),
domainProject(Obs, Atom, RealBodyEs, RealEs),
```

Then we test if the oracle can provide an equivalent answer, i.e., if there exists any χ such that $\chi \leq$ `RealEs` and $\chi \not\leq$ `IntendedEs` (an incorrectness bug generated by the selected clause).

```
domainSingleton(Obs, RealE, RealEs),
not(
    (observableAnswers(Obs, Atom, IntendedEs),
    domainSingleton(Obs, RealE, IntendedEs))    ).
```

`uncoveredE/2`. Dually to the previous case, we have to ask the oracle an answer (`IntendedE`) for a pure atomic goal

```
userdefined(Atom),
observableAnswers(Obs, Atom, IntendedEs),
domainSingleton(Obs, IntendedE, IntendedEs),
```

Then we test if the oracle simulation can provide an equivalent answer, i.e., if there exists any $\mathtt{x}$ such that $\mathtt{x} \leq$ `IntendedEs` and $\mathtt{x} \not\leq$ `RealEs` (an incompleteness bug). We perform another oracle simulation, but we have to use the entire program instead of a single clause. Hence the (partial) results for all the clauses must be joined, before we can check whether $\mathtt{x} \leq$ `RealEs`.

```
not( (
  findall(
    (R, ClauseE, Body),
    (observableClause(Obs, Atom, ClauseE, Body, _),R=Atom),
    Bodies),
  joinTheAnswers(Obs,  Atom, Bodies, RealEs),
  domainSingleton(Obs, IntendedE, RealEs)   ) ).
```

It is worth noting that meta-interpreters work with the PROLOG representation (rather than with the equational CLP representation) of the programs and (whenever possible) of the specifications.

In Section 5.8 we show the implementation of the modules for the complete observable *computed answers* and for the approximate observables *depth*($\mathtt{k}$) *answers* (Section 4.4.2) and $\mathcal{POS}$ (Section 4.4.1).

5.2.4 Towards the Diagnosis Theorems

We have now a diagnosis method (detection of incorrect clauses and uncovered $\mathbb{A}$-elements), which can equivalently be implemented in a bottom-up and in a top-down way. We are left with the problem of formally establishing the properties of the diagnosis method, i.e., of proving which is the relation between incorrect clauses and uncovered $\mathbb{A}$-elements on one side, and correctness and completeness, on the other side.

It is worth noting that correctness and completeness are defined in terms of $\alpha(\mathcal{F}[\![\mathsf{P}]\!])$, i.e., in terms of abstraction of the concrete semantics. On the other hand, incorrect clauses and uncovered $\mathbb{A}$-elements are defined directly in terms of abstract computations (the abstract immediate consequence operator $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ in the bottom-up characterization and the abstract oracle simulation in the top-down characterization). The issue of the precision of the abstract semantics becomes therefore relevant in establishing the relation between the two concepts. This is why the results for complete and approximate observables are different and will be shown separately. In particular, we have weaker results for approximate observables, because of the approximation of abstract computations.

We will first consider the case of complete observables in Section 5.3. We will later consider approximate observables in Section 5.5.

5.3 Abstract Diagnosis w.r.t. Complete Observables

Remember first that computed answers, correct answers and ground correct answers are all complete observables. Hence, the results of this section apply to the diagnosis w.r.t. computed answers [26] and to the declarative diagnoses in [91, 48]. In this section α will always denote a complete observable.

The first theorem shows the relation between partial correctness (Point 1 of Definition 5.2.1) and absence of incorrect clauses (Definition 5.2.5).

Theorem 5.3.1 *If there are no incorrect clauses in* P*, then* P *is partially correct w.r.t.* α*. The converse does not hold.*

Proof. By hypothesis, $\forall c \in \mathsf{P}.\ \mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}_\alpha) \leq \mathcal{I}_\alpha$. Hence $\mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha) \leq \mathcal{I}_\alpha$, i.e., $\mathcal{I}_\alpha$ is a pre-fixpoint of $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$. Since $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathit{lfp}\, \mathcal{P}_\alpha[\![\mathsf{P}]\!]$ (see Section 4.2.1), by Tarski's theorem $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) \leq \mathcal{I}_\alpha$.

Conversely, let $\mathsf{P} = \{p(s(x), x) \leftarrow r(x, x).\}$ and let ν be the complete observable "correct answers" (Section 4.3.1). Consider the specification

$$\mathcal{I}_\nu := \begin{cases} p(x, y) \mapsto \emptyset \\ r(x, y) \mapsto \{x = 0, y = 0\}. \end{cases}$$

P is partially correct w.r.t. $\mathcal{I}_\nu$ since $\nu(\mathcal{F}[\![\mathsf{P}]\!])(p(x, y)) = \nu(\mathcal{F}[\![\mathsf{P}]\!])(r(x, y)) = \emptyset \leq \mathcal{I}_\nu$. However, the only clause in P is incorrect on the $\mathbb{A}$-element $\sigma = p(x, y) \mapsto \{x = s(0), y = 0\}$, since $\sigma \leq \mathcal{P}_\nu[\![\mathsf{P}]\!](\mathcal{I}_\nu)$ and $\sigma \not\leq \mathcal{I}_\nu$. ∎

Note that the second part of Theorem 5.3.1 asserts that there might be incorrect clauses even if there are no incorrectness symptoms. In other words, if we just look at the semantics of the program, some incorrectness bugs can be "hidden" (because of an incompleteness bug). However, if there are no incompleteness bugs, all the incorrect clauses identify incorrectness symptoms, as shown by the following theorem.

Theorem 5.3.2 *Let* P *be complete w.r.t.* $\mathcal{I}_\alpha$*. If there exists an incorrect clause in* P *on the* $\mathbb{A}$*-element* σ*, then* σ *is an incorrectness symptom (and therefore* P *is not partially correct).*

Proof. By completeness of P and Theorem 4.3.2, $\mathcal{I}_\alpha \leq \alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!]$. Then, by monotonicity of $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$, $\mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha) \leq \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{F}_\alpha[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!]$. Thus, $\sigma \not\leq \mathcal{I}_\alpha$ and $\sigma \leq \mathcal{P}_\alpha[\![\{\mathsf{c}\}]\!](\mathcal{I}_\alpha)$ implies $\sigma \not\leq \mathcal{I}_\alpha$ and $\sigma \leq \mathcal{F}_\alpha[\![\mathsf{P}]\!] = \alpha(\mathcal{F}[\![\mathsf{P}]\!])$. ■

As in the case of declarative debugging, handling completeness turns out to be more complex, since some incompletenesses cannot be detected by comparing $\mathcal{I}_\alpha$ and $\mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha)$. The following example shows that we cannot base the diagnosis of incompleteness on the detection of uncovered $\mathbb{A}$-elements.

Example 5.3.3

Consider the program $\mathsf{P} = \{\mathtt{p} \ \mathtt{:-}\ \mathtt{p}\}$ and the specification $\mathcal{I} = \mathtt{p} \mapsto \varepsilon$ w.r.t. the computed answer observable. Then $\mathcal{P}[\![\mathsf{P}]\!](\mathcal{I}) = \mathcal{I}$, while $\mathcal{F}[\![\mathsf{P}]\!] = \lambda \mathtt{p}(\mathbf{x}).\, \emptyset$. Hence

1. there are no uncovered $\mathbb{A}$-elements in P,
2. P is not complete w.r.t. $\mathcal{I}$ (i.e., there exists an incompleteness symptom).

This shows that loops in the program can hide uncovered $\mathbb{A}$-elements.

The problem shown by Example 5.3.3 is that $\mathcal{I}$ is a fixpoint of $\mathcal{P}[\![\mathsf{P}]\!]$ different from the least fixpoint. The following theorem shows that the diagnosis of incompleteness can be based on Definition 5.2.6, if the operator $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ has a unique fixpoint.

Theorem 5.3.4 *Assume $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ has a unique fixpoint. If there are no uncovered $\mathbb{A}$-elements, then P is complete w.r.t. $\mathcal{I}_\alpha$. The converse does not hold.*

Proof. Absence of uncovered $\mathbb{A}$-elements implies $\mathcal{I}_\alpha \leq \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha)$. Hence, $\mathcal{I}_\alpha$ is a post-fixpoint of $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ and, by Tarski's theorem, $\mathcal{I}_\alpha \leq \mathit{gfp}(\mathcal{P}_\alpha[\![\mathsf{P}]\!])$. Since, by Theorem 4.3.2, $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{F}_\alpha[\![\mathsf{P}]\!] = \mathit{lfp}\, \mathcal{P}_\alpha[\![\mathsf{P}]\!]$ and, by hypothesis, $\mathit{gfp}(\mathcal{P}_\alpha[\![\mathsf{P}]\!]) = \mathit{lfp}(\mathcal{P}_\alpha[\![\mathsf{P}]\!])$, the program P is complete.

The converse does not hold, as shown by Example 5.2.7. ■

Note that, if $\mathcal{P}[\![\mathsf{P}]\!]$ has a unique fixpoint, $\mathit{lfp}\, \mathcal{P}[\![\mathsf{P}]\!] = \mathit{gfp}\, \mathcal{P}[\![\mathsf{P}]\!]$. Hence, under this hypothesis, our incompleteness symptoms are exactly the insufficiency symptoms in [48].

Note that the second part of Theorem 5.3.4 asserts that there might be uncovered $\mathbb{A}$-elements even if there are no incompleteness symptoms. This is essentially due to the fact that, if we just look at the semantics of the program, some incompleteness bugs can be "hidden", because of an incorrectness bug (note the symmetry w.r.t. the case of incorrectness diagnosis). However, if there are no incorrectness bugs, all the uncovered $\mathbb{A}$-elements identify incompleteness symptoms. In such a case, uncovered $\mathbb{A}$-elements are meaningful, as shown by the following theorem.

Theorem 5.3.5 *Let* P *be partially correct w.r.t.* $\mathcal{I}_\alpha$*. If there exists an uncovered* $\mathbb{A}$*-element* σ*, then* σ *is an incompleteness symptom (and therefore* P *is not complete).*

Proof. By Theorem 4.3.2 and partial correctness of P, $\mathcal{F}_\alpha[\![\mathsf{P}]\!] = \alpha(\mathcal{F}[\![\mathsf{P}]\!]) \leq \mathcal{I}_\alpha$. By monotonicity of $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$, $\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{F}_\alpha[\![\mathsf{P}]\!]) \leq \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha)$. Thus, $\sigma \leq \mathcal{I}_\alpha$ and $\sigma \not\leq \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha)$ implies $\sigma \leq \mathcal{I}_\alpha$ and $\sigma \not\leq \alpha(\mathcal{F}[\![\mathsf{P}]\!])$. ■

By combining Theorems 5.3.1 and 5.3.4, we can characterize total correctness, as shown by the following corollary.

Corollary 5.3.6 *Assume* $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ *has a unique fixpoint. Then,* P *is totally correct w.r.t.* $\mathcal{I}_\alpha$ *if and only if there are no incorrect clauses and no uncovered* $\mathbb{A}$*-elements.*

5.3.1 Acceptable Programs

The requirement on $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ in the hypotheses of Theorem 5.3.4 seems to be very strong. However, this property holds for a large class of programs, i.e., for *acceptable programs* as defined in [6]. Acceptable programs are the left-terminating programs, i.e., those programs for which the *SLD*-derivations of ground goals (via the leftmost selection rule) are finite.

Definition 5.3.7 *[6] Let* P *be a program and* B_P *be its Herbrand base. A* level mapping *for* P *is a function* $|\cdot| : \mathsf{B}_\mathsf{P} \to \mathbb{N}$ *from ground atoms to natural numbers. Let* $|\cdot|$ *be a level mapping for* P *and* $\mathcal{I}$ *be a (not necessarily Herbrand) model of* P*.* P *is* acceptable *w.r.t.* $|\cdot|$ *and* $\mathcal{I}$*, if for every clause* $\mathsf{A} \leftarrow \mathsf{B}_1, \ldots, \mathsf{B}_n$ *in Ground*(P) *the following implication holds, for* $i \in [1, n]$*:*

$$\mathcal{I} \models \wedge_{j=1}^{i-1} \mathsf{B}_j \Longrightarrow |\mathsf{A}| > |\mathsf{B}_i|. \tag{5.3.1}$$

Most interesting programs are acceptable (all the pure PROLOG programs in [94] are reported in [6] to be acceptable). The same property holds for the wrong versions of acceptable programs provided that the errors do not affect the left-termination property. One relevant technical property of acceptable programs is that the ground immediate consequence operator has a unique fixpoint [6]. In the following we show that the same property holds for all the immediate consequence operators $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$, corresponding to complete observables. Before giving the formal proof of the theorem, some additional definitions and lemmata are needed.

The following definition extends the original one to the case of the domain of interpretations $\mathbb{I}_\mathbb{C}$ of our collecting semantics.

Definition 5.3.8 *A* norm *for a program* P *on* $\mathbb{I}_{\mathbb{C}}$ *is a function* $\|\cdot\| : \mathbb{I}_{\mathbb{C}} \to \mathbb{N}$ *such that for every* $\mathfrak{n}$ *the set* $\{\mathsf{C} \in \mathbb{I}_{\mathbb{C}} \mid \|\mathsf{C}\| = \mathfrak{n}\}$ *is finite. A program* P *is* $\mathbb{I}_{\mathbb{C}}$-acceptable, *if there exists a norm such that, for every* $\mathsf{c} \in \mathsf{P}$ *and all finite*[5] $\mathcal{I} \in \mathbb{I}_{\mathbb{C}}$,

$$\|\mathcal{P}[\![\{\mathsf{c}\}]\!](\mathcal{I})\| > \|\mathcal{I}\|. \tag{5.3.2}$$

Lemma 5.3.9 *Every acceptable program* P *is* $\mathbb{I}_{\mathbb{C}}$*-acceptable.*

Proof. We just need to define $\|\mathsf{C}\| := \mathit{max}\left\{\left|\{\mathsf{E}\}, \mathbf{B}\right|^* \mid \mathsf{E} \in \mathsf{C}(\mathbf{B})\right\}$, where, for any $\mathbf{G}$ and relevant[6] X, $|\mathsf{X}, \mathbf{G}|^*$ is defined as $\mathit{min}\{|\mathbf{G}\vartheta| \mid \vartheta$ is a solution of $\mathsf{X}\}$ and $|\cdot|$ is the level mapping of P (Definition 5.3.7). ■

Note that the norm $\|\cdot\|$ for P is obtained by the level mapping $|\cdot|$. We will use implicitly this norm in the following. For any complete observable α and for each $\mathfrak{n}$ we can define a "projection" function $\pi_{\mathfrak{n}}^{\alpha} = \alpha \circ \pi_{\mathfrak{n}} \circ \gamma$, where $\pi_{\mathfrak{n}}(\mathsf{C}) = \bigsqcup\{\mathsf{C}' \sqsubseteq \mathsf{C} \mid \|\mathsf{C}'\| = \mathfrak{n}\}$. The functions $\pi_{\mathfrak{n}}^{\alpha}$ are precise if α is a complete observable.

Lemma 5.3.10 *Let* P *be an acceptable program and* α *be a complete observable. Then*

$$\pi_{\mathfrak{n}}^{\alpha} \circ \mathcal{P}_{\alpha}[\![\mathsf{P}]\!] = \pi_{\mathfrak{n}}^{\alpha} \circ \mathcal{P}_{\alpha}[\![\mathsf{P}]\!] \circ (\widetilde{\bigsqcup_{i<n}} \pi_i^{\alpha}). \tag{5.3.3}$$

Proof. For every $\mathfrak{n}$, every $\mathcal{I}$ and for all $\mathfrak{m} \geq \mathfrak{n}$, the functions $(\pi_{\mathfrak{n}} \circ \mathcal{P}[\![\mathsf{P}]\!] \circ \pi_{\mathfrak{m}})(\mathcal{I})$ are undefined, because $\|\mathcal{P}[\![\mathsf{P}]\!](\mathcal{I})\| > \|\mathcal{I}\|$ by hypothesis. Thus

$$\pi_{\mathfrak{n}} \circ \mathcal{P}[\![\mathsf{P}]\!] = \pi_{\mathfrak{n}} \circ \mathcal{P}[\![\mathsf{P}]\!] \circ \bigsqcup_{i<n} \pi_i. \tag{1}$$

Then the following equivalences hold.

$$\begin{aligned}
&\pi_{\mathfrak{n}}^{\alpha} \circ \mathcal{P}_{\alpha}[\![\mathsf{P}]\!] = && [\text{by definition of } \mathcal{P}_{\alpha}[\![\cdot]\!]]\\
&\pi_{\mathfrak{n}}^{\alpha} \circ \alpha \circ \mathcal{P}[\![\mathsf{P}]\!] \circ \gamma = && [\text{by } \pi_{\mathfrak{n}}^{\alpha} \text{ precision}]\\
&\alpha \circ \pi_{\mathfrak{n}} \circ \mathcal{P}[\![\mathsf{P}]\!] \circ \gamma = && [\text{by } (1)]\\
&\alpha \circ (\pi_{\mathfrak{n}} \circ \mathcal{P}[\![\mathsf{P}]\!] \circ \bigsqcup_{i<n} \pi_i) \circ \gamma = && [\text{by } \pi_{\mathfrak{n}}^{\alpha} \text{ precision}]\\
&\pi_{\mathfrak{n}}^{\alpha} \circ \alpha \circ \mathcal{P}[\![\mathsf{P}]\!] \circ \bigsqcup_{i<n} (\pi_i \circ \gamma) = && [\text{by } \mathcal{P}_{\alpha}[\![\cdot]\!] \text{ precision}]\\
&\pi_{\mathfrak{n}}^{\alpha} \circ \mathcal{P}_{\alpha}[\![\mathsf{P}]\!] \circ \alpha \circ \bigsqcup_{i<n} (\pi_i \circ \gamma) = && [\text{by } \alpha \text{ additivity}]
\end{aligned}$$

[5]Recall that $\mathcal{I}$ is finite if its support and image are finite.
[6]Recall that X is relevant for $\mathbf{G}$ if $\mathit{elim}(\mathsf{X}) \subseteq \mathit{var}(\mathbf{G})$.

$$\begin{array}{ll}
\pi^\alpha_n \circ \mathcal{P}_\alpha[\![P]\!] \circ \widetilde{\bigsqcup_{i<n}}(\alpha \circ \pi_i \circ \gamma) = & [\text{by } \pi^\alpha_n \text{ definition}] \\
\pi^\alpha_n \circ \mathcal{P}_\alpha[\![P]\!] \circ \widetilde{\bigsqcup_{i<n}} \pi^\alpha_i, &
\end{array}$$

which is the claim. ■

Theorem 5.3.11 *Let* P *be an acceptable program and* α *be a complete observable. Then* $\mathit{lfp}\, \mathcal{P}_\alpha[\![P]\!]$ *is the unique fixpoint of* $\mathcal{P}_\alpha[\![P]\!]$.

Proof. Clearly $\mathit{lfp}\, \mathcal{P}_\alpha[\![P]\!]$ is a fixpoint. Now assume that $\mathcal{X}$ and $\mathcal{Y}$ are fixpoints. We prove (by induction on n) that $\forall n.\, \pi^\alpha_n(\mathcal{X}) = \pi^\alpha_n(\mathcal{Y})$.

$\underline{n = 0|}$ The following equalities hold.

$$\begin{array}{ll}
\pi^\alpha_0(\mathcal{X}) = & [\text{since } \mathcal{X} \text{ is a fixpoint}] \\
(\pi^\alpha_0 \circ \mathcal{P}_\alpha[\![P]\!])(\mathcal{X}) = & [\text{by properties of } \pi^\alpha_0] \\
(\pi^\alpha_0 \circ \mathcal{P}_\alpha[\![P]\!])(\bot_{\mathbb{A}}) = & [\text{by properties of } \pi^\alpha_0] \\
(\pi^\alpha_0 \circ \mathcal{P}_\alpha[\![P]\!])(\mathcal{Y}) = & [\text{since } \mathcal{Y} \text{ is a fixpoint}] \\
\pi^\alpha_0(\mathcal{Y}). &
\end{array}$$

$\underline{n > 0|}$ The following equalities hold.

$$\begin{array}{ll}
\pi^\alpha_n(\mathcal{X}) = & [\text{since } \mathcal{X} \text{ is a fixpoint}] \\
(\pi^\alpha_n \circ \mathcal{P}_\alpha[\![P]\!])(\mathcal{X}) = & [\text{by Lemma 5.3.10}] \\
(\pi^\alpha_n \circ \mathcal{P}_\alpha[\![P]\!] \circ (\widetilde{\bigsqcup_{i<n}} \pi^\alpha_i))(\mathcal{X}) = & [\text{by inductive hypothesis}] \\
(\pi^\alpha_n \circ \mathcal{P}_\alpha[\![P]\!] \circ (\widetilde{\bigsqcup_{i<n}} \pi^\alpha_i))(\mathcal{Y}) = & [\text{by Lemma 5.3.10}] \\
(\pi^\alpha_n \circ \mathcal{P}_\alpha[\![P]\!])(\mathcal{Y}) = & [\text{since } \mathcal{Y} \text{ is a fixpoint}] \\
\pi^\alpha_n(\mathcal{Y}). &
\end{array}$$

Now we can conclude that there cannot be two different fixpoints, since $\mathcal{X} = (\widetilde{\bigsqcup}_{n\in\mathbb{N}} \pi^\alpha_n)(\mathcal{X}) = (\widetilde{\bigsqcup}_{n\in\mathbb{N}} \pi^\alpha_n)(\mathcal{Y}) = \mathcal{Y}$. ■

Note that this result applies to declarative debugging as well, since the least Herbrand model and the atomic logical consequence semantics are based on observables which are complete.

```
c1:  ancestor(X,Y) :- parent(Z,Y), ancestor(X,Z).
c2:  ancestor(X,Y) :- parent(Y,X).

% c2 instead of ancestor(X,Y):-parent(X,Y).
% missing parent(terach,abraham).
% missing parent(abraham,isaac).
```

Figure 5.4: The wrong acceptable program of Example 5.3.12

Example 5.3.12

This example is intended to show the relation among the various concepts involved in the diagnosis. Consider the acceptable program P of Figure 5.4, which is an *ancestor* program with a wrong clause and missing database tuples.

The answers of the incorrectness meta-interpreter are

```
incorrectClause(subst, AE, C).

Any (other) answer for goal parent(_1655,_1656) wrt obs. subst ?
>parent(terach,abraham).

Any (other) answer for goal ancestor(_2060,_2061) wrt obs. subst ?
>ancestor(terach,abraham).

Any (other) answer for goal ancestor(_2060,_2061) wrt obs. subst ?
>ancestor(abraham,isaac).

Any (other) answer for goal ancestor(_2060,_2061) wrt obs. subst ?
>ancestor(terach,isaac).

Any (other) answer for goal ancestor(_2060,_2061) wrt obs. subst ?
>no.

AE = ancestor(abraham,terach)
C = ancestor(_1479,_1480):-parent(_1480,_1479) ;

Any (other) answer for goal parent(_1655,_1656) wrt obs. subst ?
>parent(abraham,isaac).

AE = ancestor(isaac,abraham)
```

```
C = ancestor(_1479,_1480):-parent(_1480,_1479) ;

Any (other) answer for goal parent(_1655,_1656) wrt obs. subst ?
>no.

no
```

As resulting from the answers of the user, the intended interpretation w.r.t. the computed answer observable is

$$\mathcal{I} := \begin{cases} \mathsf{ancestor}(x,y) \mapsto & \big\{\{x = \mathsf{terach}, y = \mathsf{abraham}\}, \\ & \{x = \mathsf{terach}, y = \mathsf{isaac}\}, \\ & \{x = \mathsf{abraham}, y = \mathsf{isaac}\}\big\}, \\ \mathsf{parent}(x,y) \quad \mapsto & \big\{\{x = \mathsf{terach}, y = \mathsf{abraham}\}, \\ & \{x = \mathsf{abraham}, y = \mathsf{isaac}\}\big\} \end{cases}$$

The diagnosis delivers the following result: the clause `c2` is incorrect on

1. $\mathsf{ancestor}(x,y) \mapsto \{x = \mathsf{abraham}, y = \mathsf{terach}\}$,
2. $\mathsf{ancestor}(x,y) \mapsto \{x = \mathsf{isaac}, y = \mathsf{abraham}\}$.

The answers of the incompleteness meta-interpreter are

```
uncoveredE(subst, AE).

AE = ancestor(abraham,isaac) ;

AE = ancestor(terach,abraham) ;

AE = parent(abraham,isaac) ;

AE = parent(terach,abraham) ;

no
```

The diagnosis delivers the following result: the following $\mathbb{A}$-elements are uncovered.

1. $\mathsf{parent}(x,y) \mapsto \{x = \mathsf{terach}, y = \mathsf{abraham}\}$,
2. $\mathsf{parent}(x,y) \mapsto \{x = \mathsf{abraham}, y = \mathsf{isaac}\}$,
3. $\mathsf{ancestor}(x,y) \mapsto \{x = \mathsf{terach}, y = \mathsf{abraham}\}$,
4. $\mathsf{ancestor}(x,y) \mapsto \{x = \mathsf{abraham}, y = \mathsf{isaac}\}$.

Note that $\mathcal{F}[\![P]\!] = \lambda p(\mathbf{x}).\emptyset$. Hence, there are no incorrectness symptoms, even if there is an incorrect clause. Note also that the $\mathbb{A}$-element $\mathsf{ancestor}(x,y) \mapsto \{x = \mathsf{terach}, y = \mathsf{isaac}\}$ is not uncovered, even if it is an incompleteness symptom.

5.3.2 Discussion on Complete Observables

It is worth noting that the above diagnosis is not effective, unless the specification of the intended behavior is finite. In fact, if this is not the case, the bottom-up diagnosis is unfeasible, since $\mathcal{I}_\alpha$ is infinite, while the top-down diagnosis is unfeasible, since the oracle may return infinite answers to some queries. Hence the results on complete observables have no practical interest. However, they are the theoretical foundation of the effective diagnosis methods, which will be discussed in Sections 5.4 and 5.5. In order to tackle the effectivity problem, we need to be able to handle finite specifications. There exist three possible solutions.

Assertions. A specification can intentionally be defined by assertions, in the style of program verification as first suggested by [41]. Due to time limitations, we will not consider assertions in this thesis. Let us just mention that the results in [72] can provide the basis for extending diagnosis to assertions.

Partial specifications. A specification is a finite subset $\mathcal{I}^+$ of the intended semantics, plus a finite subset $\mathcal{I}^-$ of the complement of the intended semantics. The resulting diagnosis technique (partial diagnosis) is discussed in Section 5.4 and can be viewed as a formalization of symptom-based declarative debuggers. Just think of $\mathcal{I}^+$ as the union of the (finite) set of incompleteness symptoms and the (finite) set of "positive" answers of the oracle. And think of $\mathcal{I}^-$ as the union of the (finite) set of incorrectness symptoms and the (finite) set of "negative" answers of the oracle.

Approximate observables. A specification is simply an abstraction of the intended semantics. If the abstraction corresponds to an approximate observable, the specification is finite and the diagnosis method is *effective*. The diagnosis w.r.t. approximate observables, described in Section 5.5, will allow us to handle program properties such as modes, types, $depth(\mathsf{k})$ approximations and groundness dependencies, which can indeed be handled by approximate observables.

It is worth noting that the intended semantics is approximated by a subset in partial diagnosis, while it is approximated by a superset (the concretization of the abstract specification) in diagnosis w.r.t. approximate observables. This will be reflected by the diagnosis theorems of Sections 5.4 and 5.5.

5.4 Partial Diagnosis

In this section we propose one solution to the effectiveness problem, by approximating the intended behavior of a program (w.r.t. a complete observable) by a (finite) partial specification. Given a complete observable α, the specification of the intended behavior $\mathcal{I}_\alpha$ is approximated by a partial specification, which is a pair $(\mathcal{I}_\alpha^+, \mathcal{I}_\alpha^-)$, where

- $\mathcal{I}_\alpha^+$ is the (positive) partial specification of the intended behavior of P w.r.t. α (all the $\mathbb{A}$-elements in $\mathcal{I}_\alpha^+$ should be computed by P),
- $\mathcal{I}_\alpha^-$ is the (negative) partial specification of the intended behavior of P w.r.t. α (none of the $\mathbb{A}$-elements in $\mathcal{I}_\alpha^-$ should be computed by P).

The (obvious) condition on $(\mathcal{I}_\alpha^+, \mathcal{I}_\alpha^-)$ is that $\mathcal{I}_\alpha^+ \widetilde{\sqcap} \mathcal{I}_\alpha^- = \bot_{\mathbb{A}}$.

Definition 5.4.1 *Let $(\mathcal{I}_\alpha^+, \mathcal{I}_\alpha^-)$ be a partial specification. A specification $\mathcal{I}_\alpha$ is* consistent with $(\mathcal{I}_\alpha^+, \mathcal{I}_\alpha^-)$ *if $\mathcal{I}_\alpha^+ \leq \mathcal{I}_\alpha$ and $\mathcal{I}_\alpha^- \widetilde{\sqcap} \mathcal{I}_\alpha = \bot_{\mathbb{A}}$.*

The idea is that a partial specification $(\mathcal{I}_\alpha^+, \mathcal{I}_\alpha^-)$ stands for all possible specifications $\mathcal{I}_\alpha$ which are consistent with $(\mathcal{I}_\alpha^+, \mathcal{I}_\alpha^-)$.

Remark 5.4.2 Positive and negative specifications have been used in [49], for the correct answer observable ν, with the aim of separately modeling the behavior w.r.t. incorrectness and incompleteness. However, $\mathcal{I}_\nu^+$ and $\mathcal{I}_\nu^-$ are not partial specifications, rather they are specifications of the (complete) intended $\mathit{lfp}(\mathcal{P}_\nu[\![\mathsf{P}]\!])$ and of the (complete) intended complement of $\mathit{gfp}(\mathcal{P}_\nu[\![\mathsf{P}]\!])$. The derived definitions and results are completely different from ours. In particular, the complement of $\mathcal{I}_\nu^-$ is used for completeness and $\mathcal{I}_\nu^+$ is used for correctness. ■

The following definitions, given in terms of the $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ operator, generalize the definitions of incorrect clause and uncovered $\mathbb{A}$-element to the case of partial specifications.

Definition 5.4.3 *Let P be a program. If there exists an $\mathbb{A}$-element σ such that $\sigma \leq \mathcal{I}_\alpha^-$ and $\sigma \leq \mathcal{P}_\alpha[\![\{\mathsf{c}\}]\!](\mathcal{I}_\alpha^+)$, then the clause $\mathsf{c} \in \mathsf{P}$ is* p-incorrect *on σ.*

Definition 5.4.4 *Let P be a program. An $\mathbb{A}$-element σ is* p-uncovered *if $\sigma \leq \mathcal{I}_\alpha^+$ and $\sigma \not\leq \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha^+)$.*

The following lemma is a straightforward consequence of Definition 5.4.3. It asserts that p-incorrect clauses are meaningful.

Lemma 5.4.5 *If a clause c is p-incorrect on σ, then c is incorrect on σ w.r.t. any specification consistent with $(\mathcal{I}_\alpha^+, \mathcal{I}_\alpha^-)$.*

From the previous lemma and Theorem 5.3.2 we have

Theorem 5.4.6 *Let* P *be a program and* $(\mathcal{I}^+_\alpha, \mathcal{I}^-_\alpha)$ *be a partial specification w.r.t. a complete observable* α*. Assume that* P *is complete w.r.t. a specification* $\mathcal{I}_\alpha$ *consistent with* $(\mathcal{I}^+_\alpha, \mathcal{I}^-_\alpha)$*. If a clause* $\mathsf{c} \in \mathsf{P}$ *is p-incorrect on* σ*, then* σ *is an incorrectness symptom.*

It is worth noting that the diagnosis of partial correctness w.r.t. any specification $\mathcal{I}_\alpha$ consistent with $(\mathcal{I}^+_\alpha, \mathcal{I}^-_\alpha)$ cannot always be based on the detection of p-incorrect clauses, as shown by the following example.

Example 5.4.7

Consider the program in Figure 5.4 and the computed answer observable. Assume that the following partial specification is given

$$\mathcal{I}^+ := \begin{cases} \mathsf{ancestor}(\mathsf{x},\mathsf{y}) \mapsto \{\{\mathsf{x} = \mathsf{terach}, \mathsf{y} = \mathsf{abraham}\}, \\ \qquad\qquad\qquad\qquad \{\mathsf{x} = \mathsf{terach}, \mathsf{y} = \mathsf{isaac}\}, \\ \qquad\qquad\qquad\qquad \{\mathsf{x} = \mathsf{abraham}, \mathsf{y} = \mathsf{isaac}\}\} \\ \mathsf{parent}(\mathsf{x},\mathsf{y}) \quad \mapsto \{\mathsf{x} = \mathsf{abraham}, \mathsf{y} = \mathsf{isaac}\} \end{cases}$$

$$\mathcal{I}^- := \mathsf{ancestor}(\mathsf{x},\mathsf{y}) \mapsto \{\{\mathsf{x} = \mathsf{abraham}, \mathsf{y} = \mathsf{terach}\}, \\ \{\mathsf{x} = \mathsf{isaac}, \mathsf{y} = \mathsf{terach}\}\}.$$

The clause $\mathsf{c2}$ is incorrect on $\sigma = \mathsf{ancestor}(\mathsf{x},\mathsf{y}) \mapsto \{\mathsf{x} = \mathsf{abraham}, \mathsf{y} = \mathsf{terach}\}$ w.r.t. the specification given in Example 5.3.12 (which is consistent with $(\mathcal{I}^+, \mathcal{I}^-)$), but it is not p-incorrect on σ.

Let us consider now the diagnosis of completeness. The following lemma shows that the diagnosis based on the detection of p-uncovered $\mathbb{A}$-elements restricted to the specification $\mathcal{I}^+_\alpha$ is sound.

Lemma 5.4.8 *Let* P *be a program and* $(\mathcal{I}^+_\alpha, \mathcal{I}^-_\alpha)$ *be its partial specification w.r.t. a complete observable* α*. If there are no p-uncovered* $\mathbb{A}$*-elements, then no* $\mathbb{A}$*-element in* $\mathcal{I}^+_\alpha$ *is uncovered w.r.t. any specification* $\mathcal{I}_\alpha$ *consistent with* $(\mathcal{I}^+_\alpha, \mathcal{I}^-_\alpha)$*.*

Proof. By hypothesis, $\mathcal{I}^+_\alpha \leq \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}^+_\alpha)$. Since $\mathcal{I}^+_\alpha \leq \mathcal{I}_\alpha$ and by monotonicity of $\mathcal{P}_\alpha[\![\cdot]\!]$, $\mathcal{I}^+_\alpha \leq \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha)$. ∎

As was the case for complete specifications, the diagnosis of completeness can be based on $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$, only if the operator $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ has a unique fixpoint. The following theorem is a direct consequence of Lemma 5.4.8 and Theorem 5.3.4.

Theorem 5.4.9 *Let* P *be a program and* $(\mathcal{I}^+_\alpha, \mathcal{I}^-_\alpha)$ *be its partial specification w.r.t. a complete observable* α*. Assume* $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ *has a unique fixpoint. If there are no p-uncovered* $\mathbb{A}$*-elements, then* P *is complete w.r.t.* $\mathcal{I}^+_\alpha$*.*

It is worth noting that the existence of a p-uncovered $\mathbb{A}$-element does not necessarily mean that there is something missing from the program. As expected, if an $\mathbb{A}$-element is uncovered w.r.t. the partial specification, it might be covered w.r.t. a consistent complete specification. In fact, an $\mathbb{A}$-element in $\mathcal{I}^+_\alpha$ might not be in $\mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}^+_\alpha)$ just because $\mathcal{I}^+_\alpha$ is partial, i.e., it cannot be derived by $\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ because some of the correct premises are missing from $\mathcal{I}^+_\alpha$. This is shown by the following example.

Example 5.4.10

Consider the program in Figure 5.4, the computed answer observable and the following partial specification

$$\mathcal{I}^+ := \begin{cases} \mathsf{ancestor}(x,y) \mapsto \{\{x = \mathsf{terach}, y = \mathsf{abraham}\}, \\ \qquad\qquad\qquad\qquad \{x = \mathsf{terach}, y = \mathsf{isaac}\}, \\ \qquad\qquad\qquad\qquad \{x = \mathsf{abraham}, y = \mathsf{isaac}\}\} \\ \mathsf{parent}(x,y) \quad \mapsto \{x = \mathsf{terach}, y = \mathsf{abraham}\} \end{cases}$$

$$\mathcal{I}^- := \mathsf{ancestor}(x,y) \mapsto \{\{x = \mathsf{abraham}, y = \mathsf{terach}\}, \\ \{x = \mathsf{isaac}, y = \mathsf{terach}\}\}.$$

The $\mathbb{A}$-element $\sigma = \mathsf{ancestor}(x,y) \mapsto \{x = \mathsf{terach}, y = \mathsf{isaac}\}$ is p-uncovered, but it is not uncovered w.r.t. the complete specification given in Example 5.3.12 (which is consistent with $(\mathcal{I}^+, \mathcal{I}^-)$).

The meta-interpreters for partial diagnosis (Figure 5.5) can be obtained from those of Section 5.2.3, by replacing the calls to the oracle in (the definitions of the predicates) `incorrectClause/3` and `uncoveredE/2` by $\mathbb{A}$-elements of $\mathcal{I}^+$ (incompleteness symptoms and positive oracle answers in symptom-driven diagnosis) and instead of testing if the (positive) oracle can *not* provide an equivalent answer, we query the negative oracle $\mathcal{I}^-$ (incorrectness symptoms in symptom-driven diagnosis) to check whether it *can* provide an equivalent answer.

Example 5.4.11

Consider the program P of Figure 5.6, which is a "reverse" program, where the clause `c4` is wrong, w.r.t. the computed answer observable. The answers of the meta-interpreters are

```
incorrectClause(partial(Obs), RealElem, Clause) :-
    userdefined(partial(Obs), Atom),
    observableClause(Obs, Atom, ClauseE, Body, Clause),
    meetThePosAnswers(Obs, Body, ClauseE, RealBodyEs),
    domainProject(Obs, Atom, RealBodyEs, RealEs),
    domainSingleton(Obs, RealE, RealEs),
    observableNegAnswers(Obs, Atom, IntendedNegEs),
    domainSingleton(Obs, RealE, IntendedNegEs),
    showAelement(Obs, Atom, RealE, RealElem).

uncoveredE(partial(Obs), Mod, IntendedElem) :-
    userdefined(partial(Obs), Atom),
    observablePosAnswers(Obs, Atom, IntendedPosEs),
    domainSingleton(Obs, IntendedE, IntendedPosEs),
    not( (
        findall((Remember, ClauseE, Body),
            (observableClause(Obs, Atom, ClauseE, Body, _),
            Remember=Atom), Bodies),
        joinThePosAnswers(Obs,  Atom, Bodies, RealEs),
        domainSingleton(Obs, IntendedE, RealEs)   ) ),
    showAelement(Obs, Atom, IntendedE, IntendedElem).

meetThePosAnswers(Obs, Body, ClauseE, RealBodyEs) :-
    meetTheAnswers(Obs, Body, ClauseE, RealBodyEs).
observableNegAnswers(Obs, Atom, NegEs) :-
    observableAnswers(Obs, negative(Atom), Es),Es=negative(NegEs).
observablePosAnswers(Obs, Atom, Es) :-
    observableAnswers(Obs, Atom, Es).
joinThePosAnswers(Obs,  Atom, Bodies, RealEs) :-
    joinTheAnswers(Obs,  Atom, Bodies, RealEs).
```

Figure 5.5: The main module of the partial diagnosis program

```
c1:  rev([],[]).
c2:  rev([X|Xs],Ys) :- rev(Xs,Zs), append(Zs,[X],Ys).
c3:  append([],Xs,Xs).
c4:  append([X|Xs],Ys,Zs) :- append(Xs,Ys,Zs).

% c4 instead of append([X|Xs],Ys,[X|Zs]):-append(Xs,Ys,Zs).
```

Figure 5.6: The wrong reverse program of Example 5.4.11

```
incorrectClause(partial(subst), AE, C).

Any (other) answer for goal negative(rev(_2721,_2722)) wrt obs.
subst ?
>no.

Any (other) answer for goal rev(_3047,_3048) wrt obs. subst ?
>rev([V],[V]).

Any (other) answer for goal app(_3721,_3722,_3723) wrt obs. subst ?
>app([],X,X).

Any (other) answer for goal app(_3721,_3722,_3723) wrt obs. subst ?
>no.

Any (other) answer for goal rev(_3047,_3048) wrt obs. subst ?
>rev([],[]).

Any (other) answer for goal rev(_3047,_3048) wrt obs. subst ?
>no.

Any (other) answer for goal negative(app(_2772,_2773,_2774)) wrt obs.
subst ?
>negative(app([V],X,X)).

Any (other) answer for goal negative(app(_2772,_2773,_2774)) wrt obs.
subst ?
>no.
```

```
AE = app([_3106],_3026,_3026)
C = app([_2643|_2644],_2611,_2612):-app(_2644,_2611,_2612) ;

no

uncoveredE(partial(subst), AE).

no
```

As resulting from the answers of the user, the partial specification is

$$\mathcal{I}^+ := \begin{cases} \mathtt{rev}(x,y) & \mapsto \{\{x=[\,],y=[\,]\},\{x=[v],y=[v]\}\} \\ \mathtt{append}(x,y,z) & \mapsto \{x=[\,],y=z\} \end{cases}$$
$$\mathcal{I}^- := \mathtt{append}(x,y,z) \mapsto \{x=[v],y=z\}.$$

The diagnosis delivers the following results. There are no p-uncovered $\mathbb{A}$-elements, hence the program is complete w.r.t. $\mathcal{I}^+_\alpha$. The clause `c4` is p-incorrect (and then incorrect) on $\mathtt{append}(x,y,z) \mapsto \{x=[v],y=z\}$.

It is easy to prove that if the partial specification is indeed complete, we obtain exactly the results of Section 5.3.

5.4.1 Discussion on Partial Diagnosis

The overall results for partial diagnosis are:

- p-incorrect clauses always correspond to errors, w.r.t. any $\mathcal{I}_\alpha$ consistent with $(\mathcal{I}^+_\alpha, \mathcal{I}^-_\alpha)$,
- if there are no p-uncovered $\mathbb{A}$-elements (and if $\mathcal{P}_\alpha[\![P]\!]$ has a unique fixpoint) no $\mathbb{A}$-element of $\mathcal{I}^+_\alpha$ is an incompleteness symptom.

The results are weaker than those of Section 5.3, because absence of p-incorrect clauses does not imply partial correctness and p-uncovered $\mathbb{A}$-elements do not necessarily correspond to incompleteness errors.

As already mentioned, these results can be applied to practical declarative debuggers, where (some) errors are detected starting from a finite set of incorrectness and incompleteness symptoms. The first result justifies the process of determining incorrect clauses from incorrectness symptoms and tells us that incorrect clauses do always correspond to errors. Symptom-directed debuggers are of course not concerned with the problem of deciding partial correctness.

On the other side, symptom-directed debuggers derive p-uncovered $\mathbb{A}$-elements starting from an incompleteness symptom. Our negative result on p-uncovered $\mathbb{A}$-elements shows that the uncovered $\mathbb{A}$-element does not always correspond to an incompleteness error, unless the oracle can return infinite answers.

5.5 Abstract Diagnosis w.r.t. Approximate Observables

We finally turn to the case of approximate observables, which can be used both to make the diagnosis effective, as in the case of *depth*(k)-answers, and for performing the diagnosis w.r.t. abstract properties, such as groundness dependencies.

Theorem 5.5.1 *Let α be an approximate observable. If there are no incorrect clauses in P, then P is partially correct w.r.t. α. The converse does not hold.*

Proof. Since α is an approximate observable (by Theorem 4.4.2) $\alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{F}_\alpha[\![P]\!]$. From the first part of Theorem 5.3.1 it follows that, if there are no incorrect clauses in P, then $\mathcal{F}_\alpha[\![P]\!] \leq \mathcal{I}_\alpha$. Hence P turns out to be partially correct.

Conversely, let $P := \{p(s(x), x) \leftarrow r(x, x)\}$ and consider the following specification w.r.t. the Υ approximate observable.

$$\mathcal{I}_\Upsilon := \begin{cases} p(x, y) \mapsto \mathit{false} \\ r(x, y) \mapsto x \wedge y. \end{cases}$$

Since $\Upsilon(\mathcal{F}[\![P]\!])(p(x, y)) = \Upsilon(\mathcal{F}[\![P]\!])(r(x, y)) = \mathit{false} \leq \mathcal{I}_\Upsilon$, P is partially correct w.r.t. $\mathcal{I}_\Upsilon$. However, the only clause in P is incorrect on the $\mathbb{A}$-element $\sigma = p(x, y) \mapsto x \wedge y$, since $\sigma \leq \mathcal{P}_\Upsilon[\![P]\!](\mathcal{I}_\Upsilon)$ and $\sigma \not\leq \mathcal{I}_\Upsilon$. ■

Example 5.5.2

Consider the program of Figure 4.5 w.r.t. the Υ observable. The answers of the incorrectness meta-interpreter are

```
incorrectClause(pos(subst), AE, C).

Any (other) answer for goal p(_2499,_2500) wrt obs. pos(subst) ?
>p(X,Y),true.

Any (other) answer for goal q(_2499,_2500) wrt obs. pos(subst) ?
>q(X,Y),iff(X,Y).
```

```
Any (other) answer for goal r(_2499,_2500) wrt obs. pos(subst) ?
>r(X,Y),iff(X,Y).

Any (other) answer for goal q(_2621,_2622) wrt obs. pos(subst) ?
>no.

Any (other) answer for goal p(_2624,_2625) wrt obs. pos(subst) ?
>no.

no
```

As resulting from the answers of the user, the specification is

$$\mathcal{I}_\Upsilon := \begin{cases} \mathsf{p}(\mathsf{x},\mathsf{y}) \mapsto \mathit{true} \\ \mathsf{q}(\mathsf{x},\mathsf{y}) \mapsto \mathsf{x} \leftrightarrow \mathsf{y} \\ \mathsf{r}(\mathsf{x},\mathsf{y}) \mapsto \mathsf{x} \leftrightarrow \mathsf{y}. \end{cases}$$

There are no incorrect clauses. Then (by applying Theorem 5.5.1) P turns out to be partially correct w.r.t. $\mathcal{I}_\Upsilon$.

In the case of complete observables, Theorem 5.3.2 tells us that if the program is complete, then all the incorrect clauses do indeed identify incorrectness bugs. This result does not hold for approximate observables, because the abstract immediate consequence operator is in general not precise. This is shown by the following example.

Example 5.5.3

Consider the program P of Figure 5.7 and the following abstract specification w.r.t. the Υ observable. The answers of the diagnosis meta-interpreter are

```
incorrectClause(pos(subst), AE, C).

Any (other) answer for goal q(_1583) wrt obs. pos(subst) ?
>q(X),X.

Any (other) answer for goal p(_1503) wrt obs. pos(subst) ?
>p(X),X.

Any (other) answer for goal r(_1503) wrt obs. pos(subst) ?
>r(X),false.
```

```
c1:  p(f(X)) :- q(X).
c2:  q(a).
c3:  r(X) :- p(g(X)).
c4:  s(X,Y) :- r(X).
c5:  s(X,a).
```

Figure 5.7: The program of Example 5.5.3 (see also Figure 3.3)

```
Any (other) answer for goal r(_1503) wrt obs. pos(subst) ?
>no.

AE = r(_1503),and([_1503])
C = r(_1629):-p(g(_1629)) ;

Any (other) answer for goal s(_1503,_1504) wrt obs. pos(subst) ?
>s(X,Y),Y.

no

uncoveredE(pos(subst), AE).

no
```

As resulting from the answers of the user, the specification is

$$\mathcal{I}_\gamma := \begin{cases} s(x,y) \mapsto y \\ q(x) \quad \mapsto x \\ r(x) \quad \mapsto \mathit{false} \\ p(x) \quad \mapsto x. \end{cases}$$

The abstract semantics is

$$\mathcal{F}_\gamma[\![P]\!] = \begin{cases} s(x,y) \mapsto x \vee y \\ q(x) \quad \mapsto x \\ r(x) \quad \mapsto x \\ p(x) \quad \mapsto x, \end{cases}$$

while the abstraction of the concrete denotational semantics is

$$\Upsilon(\mathcal{F}[\![P]\!]) = \begin{cases} s(x,y) \mapsto y \\ q(x) \quad \mapsto x \\ r(x) \quad \mapsto \mathit{false} \\ p(x) \quad \mapsto x. \end{cases}$$

The abstract semantics is not precise, i.e., $\mathcal{F}_\Upsilon[\![P]\!] \neq \Upsilon(\mathcal{F}[\![P]\!])$. The program P turns out to be totally correct w.r.t. $\mathcal{I}_\Upsilon$. However the clause $c3$ turns out to be incorrect on the $\mathbb{A}$-element $\sigma = r(x) \mapsto x$, since $\mathcal{P}_\Upsilon[\![\{c3\}]\!](\mathcal{I}_\Upsilon)(r(x)) = x$. This is due to the approximation introduced by the $\mathcal{P}_\Upsilon[\![\cdot]\!]$ operator.

The above example shows that incorrect clauses are in general just a hint about a possible source of errors.

Once an incorrect clause is detected, one has to check on the abstraction of the concrete semantics if there is indeed a bug. This is often the case as shown by the following example.

Example 5.5.4

Consider the program P of Figure 4.1 (on page 171) and the specification $\mathcal{I}_\Upsilon := \mathtt{sum}(x,y,z) \mapsto x \wedge y \wedge z$ w.r.t. the Υ observable. We expect all the arguments of the `sum` relation to be ground. The abstraction of the concrete denotation of P is

$$\Upsilon(\mathcal{F}[\![P]\!]) = \mathtt{sum}(x,y,z) \mapsto x \wedge y \leftrightarrow z.$$

The answers of the incorrectness meta-interpreter are

```
incorrectClause(pos(subst), AE, C).

Any (other) answer for goal sum(_1981,_1982,_1983) wrt obs.
pos(subst) ?
>sum(X,Y,Z),and(X,and(Y,Z)).

Any (other) answer for goal sum(_1981,_1982,_1983) wrt obs.
pos(subst) ?
>no.

AE = sum(_1981,_1982,_1983),and([-_1983,-_1982,_1981])
C = sum(0,_2160,_2160):-true ;

no
```

```
uncoveredE(pos(subst), AE).

no
```

The abstract diagnosis shows that `c1` is an incorrect clause and that, therefore, there might be a bug, as it is actually the case, since the second and the third argument are not necessarily ground.

The relevant fact, however, is that all the clauses in the program which are actually wrong, turn out to be incorrect clauses. This can easily be proved by using the following definition of *actually-incorrect clause*. Actually-incorrect clauses are defined by using the abstraction of the concrete immediate consequence operator. As a consequence, the relation between actually-incorrect clauses and symptoms, is exactly the one between incorrect clauses and symptoms for complete observables. Namely, actually-incorrect clauses always correspond to incorrectness errors.

Definition 5.5.5 *Let* P *be a program,* $\mathcal{I}$ *be a concrete specification and* $\mathcal{I}_\alpha$ *be the corresponding abstract specification (i.e.,* $\mathcal{I}_\alpha = \alpha(\mathcal{I})$*). If there exists an* $\mathbb{A}$*-element* σ *such that* $\sigma \not\leq \mathcal{I}_\alpha$ *and* $\sigma \leq \alpha(\mathcal{P}[\![\{\mathsf{c}\}]\!](\mathcal{I}))$*, then the clause* $\mathsf{c} \in \mathsf{P}$ *is* actually-incorrect *on* σ*.*

The following theorem shows that if the program has an actually-incorrect clause it is also an incorrect clause.

Theorem 5.5.6 *Any actually-incorrect clause is an incorrect clause.*

Proof. The following hold.

$$\begin{array}{ll} \alpha(\mathcal{P}[\![\{\mathsf{c}\}]\!](\mathcal{I})) \leq & [\,\text{since } \mathcal{I} \leq \gamma(\mathcal{I}_\alpha)\,] \\ \alpha(\mathcal{P}[\![\{\mathsf{c}\}]\!](\gamma(\mathcal{I}_\alpha))) \leq & [\,\text{by Point 4 of Theorem 4.4.2}\,] \\ \mathcal{P}_\alpha[\![\{\mathsf{c}\}]\!](\mathcal{I}_\alpha). & \end{array}$$

Now, if c is actually-incorrect on σ (i.e., $\sigma \not\leq \mathcal{I}_\alpha$ and $\sigma \leq \alpha(\mathcal{P}[\![\{\mathsf{c}\}]\!](\mathcal{I}))$), then c is incorrect on σ (since $\sigma \not\leq \mathcal{I}_\alpha$ and $\sigma \leq \mathcal{P}_\alpha[\![\{\mathsf{c}\}]\!](\mathcal{I}_\alpha)$). ■

In the case of approximate observables we can no longer base the diagnosis of incompleteness on the detection of uncovered $\mathbb{A}$-elements. In fact, the absence of uncovered $\mathbb{A}$-elements, even under the unique fixpoint assumption, does not imply program completeness. Incompleteness bugs might be hidden by the approximation of the abstract semantics.

Uncovered $\mathbb{A}$-elements within a partially correct program are instead meaningful even in the case of approximate observables, as shown by the following theorem.

Theorem 5.5.7 *Let α be an approximate observable and P be partially correct w.r.t. $\mathcal{I}_\alpha$. If there exists an uncovered $\mathbb{A}$-element σ, then σ is an incompleteness symptom (and therefore P is not complete).*

Proof. The following facts hold.

$$
\begin{aligned}
&\alpha(\mathcal{F}[\![\mathsf{P}]\!]) = \\
&\qquad [\,\text{since } \mathcal{F}[\![\mathsf{P}]\!] \text{ is a fixpoint}\,] \\
&\alpha(\mathcal{P}[\![\mathsf{P}]\!](\mathcal{F}[\![\mathsf{P}]\!])) \leq \\
&\qquad [\,\text{by } \mathcal{P}_\alpha[\![\cdot]\!] \text{ correctness}\,] \\
&\mathcal{P}_\alpha[\![\mathsf{P}]\!](\alpha(\mathcal{F}[\![\mathsf{P}]\!])) \leq \\
&\qquad [\,\text{by monotonicity of } \mathcal{P}_\alpha[\![\mathsf{P}]\!] \text{ and by partial correctess of } \mathsf{P}\,] \\
&\mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha).
\end{aligned}
$$

Now, if σ is an uncovered $\mathbb{A}$-element (i.e., $\sigma \leq \mathcal{I}_\alpha$ and $\sigma \not\leq \mathcal{P}_\alpha[\![\mathsf{P}]\!](\mathcal{I}_\alpha)$), then $\sigma \not\leq \alpha(\mathcal{F}[\![\mathsf{P}]\!])$, i.e., it is an incompleteness symptom. ■

Example 5.5.8

Consider the program of Figure 4.5 and the intended specification of Example 5.5.2 w.r.t. the Υ observable. The answers of the incompleteness meta-interpreter are

```
uncoveredE(pos(subst), AE).

Any (other) answer for goal p(_1491,_1492) wrt obs. pos(subst) ?
>p(X,Y),true.

AE = p(_1491,_1492),and([]) ;

Any (other) answer for goal p(_1491,_1492) wrt obs. pos(subst) ?
>no.

Any (other) answer for goal q(_1491,_1492) wrt obs. pos(subst) ?
>q(X,Y),iff(X,Y).

Any (other) answer for goal q(_1491,_1492) wrt obs. pos(subst) ?
>no.

Any (other) answer for goal r(_1491,_1492) wrt obs. pos(subst) ?
>r(X,Y),iff(X,Y).
```

```
Any (other) answer for goal r(_1491,_1492) wrt obs. pos(subst) ?
>no.

no
```

The incompleteness diagnosis delivers the following results.

1. The $\mathbb{A}$-element $\sigma_1 = p(x, y) \mapsto \mathit{true}$ is uncovered and therefore it is an incompleteness symptom.

2. The $\mathbb{A}$-element $\sigma_2 = r(x, y) \mapsto x \leftrightarrow y$ is also an incompleteness symptom but it is not uncovered.

The incorrectness diagnosis is shown in Example 5.5.2.

We will now show an example of how the diagnosis w.r.t. $\mathit{depth}(k)$ answers can be used to approximate the (unfeasible) diagnosis w.r.t. computed answers.

Example 5.5.9

The program in Figure 5.8 is a wrong version of an automaton which recognizes the language $L = \{(ab)^n \mid n \geq 0\} \cup \{(ab)^n a \mid n \geq 0\}$. Let us consider it w.r.t. the $\mathit{depth}(2)$-answer observable (τ_2). The answers of the meta-interpreters are

```
uncoveredE(depth(2), AE).

Any (other) answer for goal acc(_2631) wrt obs. depth(2) ?
>acc([]).

Any (other) answer for goal accept(_2881) wrt obs. depth(2) ?
>accept([]).

Any (other) answer for goal accept(_2881) wrt obs. depth(2) ?
>accept([a]).

Any (other) answer for goal accept(_2881) wrt obs. depth(2) ?
>accept([a,B|J]).

Any (other) answer for goal accept(_2881) wrt obs. depth(2) ?
>no.

AE = acc([]) ;
```

```
c1:  accept([a|Xs]) :- acc(Xs).
c2:  acc([b|Xs]) :- accept(Xs).
c3:  accept([]).

% missing acc([]).
```

Figure 5.8: The program of Example 5.5.9

```
Any (other) answer for goal acc(_2631) wrt obs. depth(2) ?
>acc([b]).

Any (other) answer for goal acc(_2631) wrt obs. depth(2) ?
>acc([b,A|J]).

Any (other) answer for goal acc(_2631) wrt obs. depth(2) ?
>no.

no

incorrectClause(depth(2), AE, C).

no
```

As resulting from the answers of the user, the specification is

$$\mathcal{I}_{\tau_2} := \begin{cases} \mathsf{accept}(X) \mapsto \{\{X = []\}, \{X = [a]\}, \{X = [a, \hat{\beta}|\hat{X}]\}\} \\ \mathsf{acc}(X) \quad\;\; \mapsto \{\{X = []\}, \{X = [b]\}, \{X = [b, \hat{\alpha}|\hat{X}]\}\}. \end{cases}$$

Hence we find out that the $\mathbb{A}$-element $\mathsf{acc}(x) \mapsto \{x = []\}$ is uncovered and that P is partially correct.

Let us finally note that several interesting abstract program properties, such as modes and various notions of types (including polymorphic types) can be handled by approximate observables.

5.5.1 Discussion on Approximate Observables

The diagnosis w.r.t. approximate observables is always effective, because the abstract specification is finite. As expected, the results are weaker than those of complete

observables, just because of approximation. Namely,

- absence of incorrect clauses implies partial correctness,
- every incorrectness error is identified by an incorrect clause. However an incorrect clause does not always correspond to a bug,
- uncovered $\mathbb{A}$-elements always correspond to incompleteness bugs.
- there exists no sufficient condition for completeness.

The results, even if weaker, are useful and comparable to those obtained by verification techniques (see, for example, [5, 4]). In fact, if we consider the case where specifications consist of post-conditions only, both abstract diagnosis and verification provide *a sufficient condition for partial correctness*, which is well-assertedness in the case of verification and absence of incorrect clauses in abstract diagnosis. For both techniques there is no sufficient condition for completeness. In order to verify completeness, we have to rely on a fixpoint (the model of a transformed program or the abstraction of the concrete semantics), which, in general, cannot be computed in a finite number of steps. As expected, abstract diagnosis (whose aim is locating bugs rather than just proving correctness) gives us also information useful for debugging, by means of incorrect clauses and uncovered $\mathbb{A}$-elements.

Let us finally note that approximate observables can have all the properties of compete observables, for suitable classes of programs. A characterization of these classes of programs, would make the stronger results for complete observables applicable (in an effective way). Namely, we would obtain a sufficient condition for completeness and all the incorrect clauses would indeed correspond to incorrectness bugs.

As already mentioned, the approximation given by approximate observables is somewhat dual of the one given by partial diagnosis. This is reflected by the duality of the results. In fact, in partial diagnosis we have a sufficient condition for completeness instead of a sufficient condition for partial correctness. Moreover, p-incorrect clauses always correspond to bugs, while this is not the case for p-uncovered $\mathbb{A}$-elements.

This suggests that we might improve our results on completeness by taking an under-approximation as abstraction. Of course, if we want to preserve the results on partial correctness, we need to keep also the standard over-approximation. This would lead us to a theory of abstract diagnosis, where two different abstractions are used at the same time. This would in turn require a specification consisting of a pair of abstractions of the intended concrete semantics. This is an interesting idea for further research that might build upon the already mentioned approach by Ferrand [49], which works with pairs of specifications.

5.6 Modular Abstract Diagnosis

In modular abstract diagnosis we are concerned with programs composed of separate modules. The idea is that of performing the diagnosis in a modular way, i.e., module by module. Modular analysis is usually based on an *OR*-compositional semantics. For example, the modular analysis framework in [18] is based on the *OR*-compositional version of the s-semantics [14]. Our concrete semantics *is not* *OR*-compositional and this is obviously true for all its abstractions. However, we can note that abstract diagnosis does not require to actually compute the abstract semantics, since it is simply based on one application of the abstract immediate consequence operator to the specification. The s-semantics immediate consequence operator is known to be *OR*-compositional (see for example [76]). The same result holds (by definition) for all the abstract immediate consequence operator corresponding to complete and approximate observables. The conclusion is that our theory of abstract diagnosis can directly be applied to modular diagnosis, as we will formally show in the following.

We assume a program P to be partitioned into *predicate-disjoint* modules [18], such that each predicate symbol is completely defined by a single module. Namely,

Definition 5.6.1 *A program partitioning* $\mathsf{P}_1, \ldots, \mathsf{P}_n$ *is* predicate-disjoint *if* $\forall i \neq j.\ \mathit{PredsDef}(\mathsf{P}_i) \cap \mathit{PredsDef}(\mathsf{P}_j) = \emptyset$*, where* $\mathit{PredsDef}(\mathsf{P}) := \{p \mid p(\mathbf{t}) \leftarrow B \in \mathsf{P}\}$.

Specifications $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$ are associated to modules $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Since the partition is predicate-disjoint, any $\mathcal{I}^i_\alpha$ is undefined for all the pure atoms with predicate *not* in $\mathit{PredsDef}(\mathsf{P}_i)$ and then $\forall i \neq j.\ \mathit{dom}(\mathcal{I}^i_\alpha) \cap \mathit{dom}(\mathcal{I}^j_\alpha) = \emptyset$. It is worth noting that our definition of partitioning does not require a *hierarchical decomposition*, since mutual recursion between modules is possible.

The overall specification is $\mathcal{I}_\alpha = \mathcal{I}^1_\alpha \,\widetilde{\sqcup} \ldots \widetilde{\sqcup}\, \mathcal{I}^n_\alpha$. A module P_i does not necessarily need to use all the other modules. Hence we introduce the operator *use* which gives those specifications which are *relevant* to a module P_i, i.e., $\mathit{use}(\mathsf{P}_i) = \big\{\mathcal{I}^j_\alpha \mid \textit{module } \mathsf{P}_i \textit{ uses (i.e., clause bodies contain) predicates which are defined by } \mathsf{P}_j\big\}$[7]. P_i is a *basic* module if it does not use other modules, i.e., if $\mathit{use}(\mathsf{P}_i) = \emptyset$.

The decomposition into modules allows us to define incorrect clauses and uncovered $\mathbb{A}$-elements in a (more efficient) modular way.

Definition 5.6.2 *Let* $\mathsf{P}_1, \ldots, \mathsf{P}_n$ *be a program partitioning and let* c *be a clause in* P_i*, for some* $1 \leq i \leq n$*. If there exists an* $\mathbb{A}$*-element* σ *such that* $\sigma \not\leq \mathcal{I}^i_\alpha$ *and* $\sigma \leq \mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}^i_\alpha \,\widetilde{\sqcup} \widetilde{\bigsqcup} \mathit{use}(\mathsf{P}_i))$*, then the clause* $c \in \mathsf{P}$ *is* m-incorrect *on* σ.

[7] Note that $\widetilde{\bigsqcup} \mathit{use}(\mathsf{P}_i) \leq \mathcal{I}_\alpha$ and that $\mathit{dom}(\widetilde{\bigsqcup} \mathit{use}(\mathsf{P}_i)) \cap \mathit{dom}(\mathcal{I}^i_\alpha) = \emptyset$.

c is m-incorrect on σ if it derives, from the (relevant part of the) intended semantics, an $\mathbb{A}$-element which is not in the module's intended semantics. Note that $\mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}^i_\alpha \widetilde{\sqcup} \widetilde{\bigsqcup} use(P_i)) = \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}_\alpha)$.

Definition 5.6.3 *Let $P_1, \ldots, P_n$ be a program partitioning and let $\sigma = p(\mathbf{x}) \mapsto D$ be an $\mathbb{A}$-element such that $p \in PredsDef(P_i)$, for some $1 \leq i \leq n$. σ is* m-uncovered *if $\sigma \leq \mathcal{I}^i_\alpha$ and $\sigma \not\leq \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}^i_\alpha \widetilde{\sqcup} \widetilde{\bigsqcup} use(P_i))$.*

An $\mathbb{A}$-element in the intended semantics of a module is m-uncovered if there are no clauses in the module deriving it from the (relevant part of the) intended semantics.

It is worth noting that, in the above definitions of m-incorrect clauses and m-uncovered $\mathbb{A}$-elements, we compare two denotations which give a meaning only to those predicates which are defined inside the module and that we are only concerned with the specifications used by the module.

The meta-interpreter implementation of the *algorithms* to determine m-incorrect clauses and m-uncovered $\mathbb{A}$-elements is shown in Figure 5.9. As usual, it is parametric w.r.t. the observable modules, which are exactly the ones of Section 5.2.3 (see Section 5.8).

The following theorems show that the general results on complete and approximate observables of abstract diagnosis do apply to modular abstract diagnosis.

Theorem 5.6.4 *Let $P = P_1 \cup \cdots \cup P_n$. If there are no m-incorrect clauses in any module P_i, then P is partially correct w.r.t. α.*

Proof. By hypothesis, for any i, $\forall c \in P_i.\ \mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}^i_\alpha \widetilde{\sqcup} \widetilde{\bigsqcup} use(P_i)) \leq \mathcal{I}^i_\alpha$. Hence $\forall i.\ \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}_\alpha) \leq \mathcal{I}^i_\alpha$. Now, by definition of $\mathcal{P}_\alpha[\![\cdot]\!]$, $\mathcal{P}_\alpha[\![\cup P_i]\!](\mathcal{I}_\alpha) \leq \widetilde{\bigsqcup} \mathcal{I}^i_\alpha = \mathcal{I}_\alpha$. Hence $\mathcal{I}_\alpha$ is a pre-fixpoint of $\mathcal{P}_\alpha[\![P]\!]$ and then, since $\alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{F}_\alpha[\![P]\!] = lfp\, \mathcal{P}_\alpha[\![P]\!]$, by Tarski's theorem $\alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{I}_\alpha$. ■

Theorem 5.6.5 *Let α be a complete observable and $P = P_1 \cup \cdots \cup P_n$ be a complete program w.r.t. $\mathcal{I}_\alpha$. If, for some i, there exists an m-incorrect clause in P_i, then P is not partially correct.*

Proof. By completeness of P and Theorem 4.3.2, $\mathcal{I}_\alpha \leq \alpha(\mathcal{F}[\![P]\!]) = \mathcal{F}_\alpha[\![P]\!]$. Then, by monotonicity of $\mathcal{P}_\alpha[\![\cdot]\!]$, $\mathcal{P}_\alpha[\![P]\!](\mathcal{I}_\alpha) \leq \mathcal{P}_\alpha[\![P]\!](\mathcal{F}_\alpha[\![P]\!]) = \mathcal{F}_\alpha[\![P]\!]$. Thus, $\sigma \not\leq \mathcal{I}^i_\alpha$ and $\sigma \leq \mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}_\alpha) = \mathcal{P}_\alpha[\![\{c\}]\!](\mathcal{I}^i_\alpha \widetilde{\sqcup} \widetilde{\bigsqcup} use(P_i))$ implies $\sigma \not\leq \mathcal{I}^i_\alpha$ and $\sigma \leq \mathcal{F}_\alpha[\![P]\!] = \alpha(\mathcal{F}[\![P]\!])$, which means that P is not partially correct. ■

Theorem 5.6.6 *Let $P = P_1 \cup \cdots \cup P_n$ be partially correct w.r.t. $\mathcal{I}_\alpha$. If there exists an m-uncovered $\mathbb{A}$-element, then P is not complete.*

```
incorrectClause(Obs, Mod, RealElem, Clause) :-
    userdefined(Mod, Atom),
    observableClause(Obs, Atom, ClauseE, Body, Clause),
    meetTheAnswers(Obs, Body, ClauseE, RealBodyEs),
    domainProject(Obs, Atom, RealBodyEs, RealEs),
    domainSingleton(Obs, RealE, RealEs),
    not(
        (observableAnswers(Obs, Atom, IntendedEs),
        domainSingleton(Obs, RealE, IntendedEs))    ),
    showAelement(Obs, Atom, RealE, RealElem).

uncoveredE(Obs, Mod, IntendedElem) :-
    userdefined(Mod, Atom),
    observableAnswers(Obs, Atom, IntendedEs),
    domainSingleton(Obs, IntendedE, IntendedEs),
    not( (
        findall(
            (R, ClauseE, Body),
            (observableClause(Obs, Atom, ClauseE, Body, _),R=Atom),
            Bodies),
        joinTheAnswers(Obs,  Atom, Bodies, RealEs),
        domainSingleton(Obs, IntendedE, RealEs)   ) ),
    showAelement(Obs, Atom, IntendedE, IntendedElem).
```

Figure 5.9: The main module of the modular diagnosis program

Proof. Let $\sigma = p(\mathbf{x}) \mapsto D$ with $p \in \mathit{PredsDef}(P_i)$, for some $1 \leq i \leq n$. By Theorem 4.3.2 and partial correctness of P, $\mathcal{F}_\alpha[\![P]\!] = \alpha(\mathcal{F}[\![P]\!]) \leq \mathcal{I}_\alpha$. By monotonicity of $\mathcal{P}_\alpha[\![P]\!]$,

$$\alpha(\mathcal{F}[\![P]\!]) = \mathcal{P}_\alpha[\![P]\!](\mathcal{F}_\alpha[\![P]\!]) \leq \mathcal{P}_\alpha[\![P]\!](\mathcal{I}_\alpha). \tag{1}$$

Now, $\sigma \leq \mathcal{I}^i_\alpha$ and $\sigma \not\leq \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}_\alpha) = \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}^i_\alpha \,\widetilde{\sqcup} \widetilde{\bigsqcup} \mathit{use}(P_i))$ implies (since the other modules cannot define predicate p) $\sigma \leq \mathcal{I}^i_\alpha$ and $\sigma \not\leq \mathcal{P}_\alpha[\![P]\!](\mathcal{I}_\alpha)$. Thus $\sigma \leq \mathcal{I}^i_\alpha \leq \mathcal{I}_\alpha$ and, by (1), $\sigma \not\leq \alpha(\mathcal{F}[\![P]\!])$, which means that P is not complete. ■

Theorem 5.6.7 *Let α be a complete observable, $P = P_1 \cup \cdots \cup P_n$ be a program and assume $\mathcal{P}_\alpha[\![P]\!]$ has a unique fixpoint. If there are no m-uncovered $\mathbb{A}$-elements, then P is complete w.r.t. $\mathcal{I}_\alpha$.*

Proof. Absence of m-uncovered $\mathbb{A}$-elements implies, for all i, $\mathcal{I}^i_\alpha \leq \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}_\alpha) = \mathcal{P}_\alpha[\![P_i]\!](\mathcal{I}^i_\alpha \,\widetilde{\sqcup} \widetilde{\bigsqcup} \mathit{use}(P_i))$. Hence, $\mathcal{I}_\alpha \leq \mathcal{P}_\alpha[\![P]\!](\mathcal{I}_\alpha)$, i.e., $\mathcal{I}_\alpha$ is a post-fixpoint of $\mathcal{P}_\alpha[\![P]\!]$. Then, by Tarski's theorem, $\mathcal{I}_\alpha \leq \mathit{gfp}(\mathcal{P}_\alpha[\![P]\!])$. Since, by Theorem 4.3.2, $\alpha(\mathcal{F}[\![P]\!]) = \mathcal{F}_\alpha[\![P]\!] = \mathit{lfp}\, \mathcal{P}_\alpha[\![P]\!]$ and, by hypothesis, $\mathit{gfp}(\mathcal{P}_\alpha[\![P]\!]) = \mathit{lfp}(\mathcal{P}_\alpha[\![P]\!])$, the program P is complete. ■

Example 5.6.8

The program in Figure 5.10 is a wrong version of a program verifying satisfiability of boolean formulas (built with *or* and *neg*) which has a missing clause. We consider the approximate observable *depth*(2)-answers (τ_2) where we replace all the subterms at depth greater than 2 by a fresh variable (see Section 4.4.2). The answers of the diagnosis meta-interpreters are

```
incorrectClause(depth(2), modSat, AE, C).

Any (other) answer for goal sat(_1710) wrt obs. depth(2) ?
>sat(or(X,true)).

Any (other) answer for goal sat(_1710) wrt obs. depth(2) ?
>sat(true).

Any (other) answer for goal sat(_2173) wrt obs. depth(2) ?
>sat(neg(false)).

Any (other) answer for goal sat(_2173) wrt obs. depth(2) ?
>sat(neg(neg(J))).
```

Module "satisfiable"

```
sat(true).
sat(or(X,Y)) :- sat(X).
sat(neg(X)) :- inv(X).

%the clause sat(or(X,Y)) :- sat(Y). is missing
```

Module "invalid", this module *is not* supplied to the diagnoser.

```
inv(false).
inv(neg(X)) :- sat(X).
inv(or(X,Y)) :- inv(X),inv(Y).
```

Figure 5.10: The program of Example 5.6.8

```
Any (other) answer for goal sat(_2173) wrt obs. depth(2) ?
>sat(or(true,Y)).

Any (other) answer for goal sat(_2197) wrt obs. depth(2) ?
>sat(neg(or(J,K))).

Any (other) answer for goal sat(_2197) wrt obs. depth(2) ?
>sat(or(Y,neg(J))).

Any (other) answer for goal sat(_2197) wrt obs. depth(2) ?
>sat(or(Y,or(J,K))).

Any (other) answer for goal sat(_2197) wrt obs. depth(2) ?
>sat(or(neg(J),Y)).

Any (other) answer for goal sat(_2197) wrt obs. depth(2) ?
>sat(or(or(J,K),Y)).

Any (other) answer for goal sat(_1830) wrt obs. depth(2) ?
>no.

Any (other) answer for goal inv(_1780) wrt obs. depth(2) ?
>inv(false).
```

```
Any (other) answer for goal inv(_1780) wrt obs. depth(2) ?
>inv(neg(neg(J))).

Any (other) answer for goal inv(_1780) wrt obs. depth(2) ?
>inv(neg(or(J,K))).

Any (other) answer for goal inv(_1780) wrt obs. depth(2) ?
>inv(neg(true)).

Any (other) answer for goal inv(_1780) wrt obs. depth(2) ?
>inv(or(false,neg(J))).

Any (other) answer for goal inv(_1780) wrt obs. depth(2) ?
>inv(or(false,or(J,K))).

Any (other) answer for goal inv(_1780) wrt obs. depth(2) ?
>inv(or(neg(J),false)).

Any (other) answer for goal inv(_1780) wrt obs. depth(2) ?
>inv(or(or(J,K),false)).

Any (other) answer for goal inv(_1780) wrt obs. depth(2) ?
>inv(or(false,false)).

Any (other) answer for goal inv(_1780) wrt obs. depth(2) ?
>no.

no

uncoveredE(depth(2), modSat, AE).

AE = sat(or(_1620,or(_1621,_1622))) ;

AE = sat(or(_1620,neg(_1621))) ;

AE = sat(or(_1620,true)) ;

no
```

As resulting from the answers of the user, the specification of the module "sat" is

$$\begin{aligned}\mathcal{I}_{\tau_2} := \mathsf{sat}(x) \mapsto \big\{&\{x = \mathsf{true}\}, \{x = \mathsf{or}(Y, \mathsf{true})\}, \{x = \mathsf{or}(\mathsf{true}, Y)\},\\ &\{x = \mathsf{or}(\mathsf{neg}(\hat{x}), Y)\}, \{x = \mathsf{or}(Y, \mathsf{neg}(\hat{x}))\},\\ &\{x = \mathsf{neg}(\mathsf{false})\}, \{x = \mathsf{neg}(\mathsf{neg}(\hat{x}))\},\\ &\{x = \mathsf{neg}(\mathsf{or}(\hat{x}, \hat{y}))\}, \{x = \mathsf{or}(Y, \mathsf{or}(\hat{x}, \hat{y}))\},\\ &\{x = \mathsf{or}(\mathsf{or}(\hat{x}, \hat{y}), Y)\}\big\}\end{aligned}$$

while the specification of the module "invalid" is

$$\begin{aligned}\mathcal{K}_{\tau_2} := \mathsf{inv}(x) \mapsto \big\{&\{x = \mathsf{false}\}, \{x = \mathsf{or}(\mathsf{false}, \mathsf{false})\}, \{x = \mathsf{neg}(\mathsf{true})\},\\ &\{x = \mathsf{neg}(\mathsf{neg}(\hat{x}))\}, \{x = \mathsf{neg}(\mathsf{or}(\hat{x}, \hat{y}))\},\\ &\{x = \mathsf{or}(\mathsf{false}, \mathsf{neg}(\hat{x}))\}, \{x = \mathsf{or}(\mathsf{neg}(\hat{x}), \mathsf{false})\},\\ &\{x = \mathsf{or}(\mathsf{false}, \mathsf{or}(\hat{x}, \hat{y}))\}, \{x = \mathsf{or}(\mathsf{or}(\hat{x}, \hat{y}), \mathsf{false})\}\big\}\end{aligned}$$

Hence we find out that there are no m-incorrect clauses and then (by Theorem 5.6.4) the program is partially correct. Furthermore, the following $\mathbb{A}$-element are m-uncovered.

1. $\mathsf{sat}(x) \mapsto \{x = \mathsf{or}(y, \mathsf{or}(v, w))\}$,
2. $\mathsf{sat}(x) \mapsto \{x = \mathsf{or}(y, \mathsf{neg}(z))\}$,
3. $\mathsf{sat}(x) \mapsto \{x = \mathsf{or}(y, \mathsf{true})\}$.

Then (by Theorem 5.6.6) the (entire) program is not complete.

The above results show that, if we split the program and the specification into modules, we can determine incorrect clauses and uncovered $\mathbb{A}$-elements by means of (more efficient) *modular* algorithms. We will now turn to the (more interesting) case where we consider a single module (all the other modules may be not implemented yet), and we want to debug it, under the assumption that all the other missing modules do satisfy their specifications (i.e., are totally correct). The diagnosis will still be based on Definitions 5.6.2 and 5.6.3. However we have to introduce a new definition of partial correctness and completeness for a single module.

The new definitions are of course given in terms of the concrete semantics of a module P_i, which can be determined from the clauses in P_i and from the concrete semantics of the (missing) modules used by P_i. Since these modules have not been implemented yet, their concrete semantics is simply the concretization of the abstract specifications. This is achieved by first defining the following concrete ($\mathcal{I}_\alpha$-augmented) immediate consequence operator:

$$\mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!](\mathcal{K}) := \mathcal{P}[\![\mathsf{P}_i]\!](\mathcal{K} \sqcup \bigsqcup \gamma(\mathit{use}(\mathsf{P}_i))).$$

This operator is continuous. The concrete semantics of a program module P_i can be then defined as

$$\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!] := \mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]\uparrow\omega\,.$$

This leads to the new notions of partial correctness and completeness of a module w.r.t. the intended abstract semantics of the whole program.

Definition 5.6.9 *Let $\mathsf{P}_1, \ldots, \mathsf{P}_n$ be a program partitioning, α be an observable, and $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$ be an intended module semantics. A module P_i is*

1. m-partially correct *w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$ if $\alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]) \leq \mathcal{I}^i_\alpha$.*
2. m-complete *w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$ if $\mathcal{I}^i_\alpha \leq \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])$*
3. P *is* m-totally correct *w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$, if $\alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]) = \mathcal{I}^i_\alpha$*

The proof of the theorems uses the following (continuous) abstract ($\mathcal{I}_\alpha$-augmented) immediate consequence operator and its fixpoint.

$$\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{K}_\alpha) := \mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{K}_\alpha \,\widetilde{\sqcup} \widetilde{\bigsqcup} \mathit{use}(\mathsf{P}_i)), \qquad \mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!] := \mathit{lfp}(\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]),$$

Theorem 5.6.10 *If there are no m-incorrect clauses in P_i, then P_i is m-partially correct w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$.*

Proof. By hypothesis, for any clause c in P_i, $\mathcal{P}_\alpha[\![\{\mathsf{c}\}]\!](\mathcal{I}^i_\alpha \,\widetilde{\sqcup} \widetilde{\bigsqcup} \mathit{use}(\mathsf{P}_i)) \leq \mathcal{I}^i_\alpha$. Hence $\mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}_\alpha) \leq \mathcal{I}^i_\alpha$. Now, by definition, $\mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}_\alpha) = \mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha)$ and then $\mathcal{I}^i_\alpha$ is a pre-fixpoint of $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$. Then, since $\alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]) \leq \mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!] = \mathit{lfp}(\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]) \leq \mathcal{I}^i_\alpha$, P_i is m-partially correct w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$. ■

Theorem 5.6.11 *Let α be a complete observable and P_i be an m-complete module w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$, for some $1 \leq i \leq n$. If there exists an m-incorrect clause in P_i, then P_i is not m-partially correct.*

Proof. By m-completeness of P_i and since α is complete, $\mathcal{I}^i_\alpha \leq \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]) = \mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$. By monotonicity of $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$, $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha) \leq \mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]) = \mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$. On the other hand, by definition, $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha) = \mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}_\alpha)$. Thus, $\sigma \not\leq \mathcal{I}^i_\alpha$ and $\sigma \leq \mathcal{P}_\alpha[\![\{\mathsf{c}\}]\!](\mathcal{I}_\alpha)$ implies $\sigma \not\leq \mathcal{I}^i_\alpha$ and $\sigma \leq \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])$, i.e., P_i is not m-partially correct. ■

Theorem 5.6.12 *Let P_i be m-partially correct w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$, for some $1 \leq i \leq n$. If there exists an m-uncovered $\mathbb{A}$-element in P_i, then P_i is not m-complete.*

Proof. The operator $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$ is correct. Indeed,

$$
\begin{array}{ll}
(\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!] \circ \alpha)(\mathcal{K}) = & [\text{by } \mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!] \text{ definition}] \\
\mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\alpha(\mathcal{K}) \widetilde{\sqcup} \widetilde{\bigsqcup} \mathit{use}(\mathsf{P}_i)) = & [\text{since } \alpha \text{ is additive and } \alpha\gamma = \mathit{Id}] \\
(\mathcal{P}_\alpha[\![\mathsf{P}_i]\!] \circ \alpha)(\mathcal{K} \sqcup \bigsqcup \gamma\, \mathit{use}(\mathsf{P}_i)) \geq & [\text{since } \mathcal{P}_\alpha[\![\mathsf{P}_i]\!] \text{ is correct}] \\
(\alpha \circ \mathcal{P}[\![\mathsf{P}_i]\!])(\mathcal{K} \sqcup \bigsqcup \gamma\, \mathit{use}(\mathsf{P}_i)) = & [\text{by } \mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!] \text{ definition}] \\
(\alpha \circ \mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])(\mathcal{K}). &
\end{array}
$$

Hence,

$$
\begin{array}{l}
\alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]) = \\
\qquad [\text{since } \mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!] \text{ is a fixpoint of } \mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]] \\
\alpha(\mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!](\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])) \leq \\
\qquad [\text{by the previous result}] \\
\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])) \leq \\
\qquad [\text{by monotonicity of } \mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!] \text{ and m-partial correctness of } \mathsf{P}_i] \\
\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha) = \\
\qquad [\text{by definition}] \\
\mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}_\alpha).
\end{array}
$$

Now, if σ is an m-uncovered $\mathbb{A}$-element (i.e., $\sigma \leq \mathcal{I}^i_\alpha$ and $\sigma \not\leq \mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha)$), then $\sigma \leq \mathcal{I}^i_\alpha$ and $\sigma \not\leq \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])$, i.e., P_i is not m-complete. ■

Theorem 5.6.13 *Let α be a complete observable and assume that $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$ has a unique fixpoint (for some $1 \leq i \leq n$). If there are no m-uncovered $\mathbb{A}$-elements in P_i, then P_i is m-complete w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$.*

Proof. Absence of m-uncovered $\mathbb{A}$-elements in P_i implies $\mathcal{I}^i_\alpha \leq \mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}_\alpha)$. Since, by definition, $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha) = \mathcal{P}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}_\alpha)$, $\mathcal{I}^i_\alpha \leq \mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!](\mathcal{I}^i_\alpha)$, i.e., $\mathcal{I}^i_\alpha$ is a post-fixpoint of $\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$ and, by Tarski's theorem, $\mathcal{I}^i_\alpha \leq \mathit{gfp}(\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!])$.

Now, by hypothesis, $\mathit{gfp}(\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]) = \mathit{lfp}(\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]) = \mathcal{F}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!] = \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])$. Thus, $\mathcal{I}^i_\alpha \leq \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!])$, i.e., P_i is m-complete w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$. ■

Let us now consider again Example 5.5.9 with this modular perspective.

Example 5.6.14

The program P_{acc} in Figure 5.11 is a wrong version of a module of an automaton which recognizes the language $\mathsf{L} = \{(ab)^n \mid n \geq 0\} \cup \{(ab)^n a \mid n \geq 0\}$. The complete program is in Figure 5.8, and the complete diagnosis is in Example 5.5.9. The answers of the meta-interpreters are

```
acc([b|Xs]) :- accept(Xs).

%the clause acc([]). is missing
```

Figure 5.11: The program of Example 5.6.14

```
uncoveredE(depth(2), modAcc, AE).

Any (other) answer for goal acc(_3430) wrt obs. depth(2) ?
>acc([]).

Any (other) answer for goal accept(_3680) wrt obs. depth(2) ?
>accept([]).

Any (other) answer for goal accept(_3680) wrt obs. depth(2) ?
>accept([a]).

Any (other) answer for goal accept(_3680) wrt obs. depth(2) ?
>accept([a,B|J]).

Any (other) answer for goal accept(_3680) wrt obs. depth(2) ?
>no.

AE = acc([]) ;

Any (other) answer for goal acc(_3430) wrt obs. depth(2) ?
>acc([b]).

Any (other) answer for goal acc(_3430) wrt obs. depth(2) ?
>acc([b,A|J]).

Any (other) answer for goal acc(_3430) wrt obs. depth(2) ?
>no.

no

incorrectClause(depth(2), modAcc, AE, C).
```

no

As resulting from the answers of the user, the specification of the diagnosed module w.r.t. the *depth*(2)-answer observable (τ_2) is

$$\mathcal{I}_{\tau_2} := \mathsf{acc}(X) \mapsto \big\{\{X = []\}, \{X = [b]\}, \{X = [b, \hat{a}|\hat{x}]\}\big\}.$$

while the specification of the other module is

$$\mathcal{K}_{\tau_2} := \mathsf{accept}(X) \mapsto \big\{\{X = []\}, \{X = [a]\}, \{X = [a, \hat{b}|\hat{x}]\}\big\}$$

Hence we find out that P_{acc} is m-partially correct (without actually computing $\mathcal{F}^{\mathcal{I}_\alpha}[\![P_{acc}]\!]$) and that the $\mathbb{A}$-element $\mathsf{acc}(x) \mapsto \{x = []\}$ is m-uncovered. Then (by Theorem 5.6.12) the module is not m-complete (and the entire program is not complete).

Theorem 5.6.15 *Let $P = P_1 \cup \cdots \cup P_n$. If all the modules P_i are m-partially correct w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$, then P is partially correct w.r.t. $\mathcal{I}_\alpha$. Moreover, if P has a unique fixpoint and all the modules P_i are m-complete w.r.t. $\mathcal{I}^1_\alpha, \ldots, \mathcal{I}^n_\alpha$, then P is complete w.r.t. $\mathcal{I}_\alpha$.*

Proof. We prove the two points separately.

partial correctness Let $\mathcal{K} = \bigsqcup_i \mathcal{K}^i$. For any i,

$$\mathcal{P}^{\mathcal{I}_\alpha}[\![P_i]\!](\mathcal{K}^i) = \mathcal{P}[\![P_i]\!](\mathcal{K}^i \sqcup \bigsqcup \gamma\, use(P_i)) \geq \mathcal{P}[\![P_i]\!](\mathcal{K}^i)$$

Thus $\bigsqcup_i \mathcal{P}^{\mathcal{I}_\alpha}[\![P_i]\!](\mathcal{K}^i) \geq \mathcal{P}[\![P]\!](\mathcal{K})$ and the following facts hold.

$$\begin{array}{ll}
\alpha(\mathcal{F}[\![P]\!]) \leq & [\text{by the previous result}] \\
\alpha(\bigsqcup_i \mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!]) = & [\text{by additivity}] \\
\widetilde{\bigsqcup_i} \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!]) \leq & [\text{by m-partial correctness}] \\
\widetilde{\bigsqcup_i} \mathcal{I}^i_\alpha = \mathcal{I}_\alpha &
\end{array}$$

completeness The following facts hold.

$$\mathcal{I}_\alpha = \widetilde{\bigsqcup_i} \mathcal{I}^i_\alpha \geq \qquad [\text{by m-partial correctness}]$$

$$\widetilde{\bigsqcup_i} \alpha(\mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!]) = \qquad [\text{by additivity}]$$

$$\alpha(\bigsqcup_i \mathcal{F}^{\mathcal{I}_\alpha}[\![P_i]\!]) \geq \qquad [\text{by the fixpoint uniqueness of } P]$$

$$\alpha(\mathcal{F}[\![P]\!])$$

■

5.7 Discussion on the Results of Abstract Diagnosis

We have introduced several effective techniques to perform the diagnosis of pure logic programs.

Partial diagnosis can be used whenever we have a (finite) partial knowledge about the intended behavior. This knowledge can be derived from symptom detection (performed using testing techniques) and symptom-directed queries to the user, as in the symptom-directed debuggers. One might think of other partial knowledge acquisition techniques.

Diagnosis w.r.t. approximate observables is instead useful when one wants to perform the diagnosis w.r.t. properties which can be modeled by abstractions over Noetherian domains. This is interesting because several useful properties are modeled by approximate observables or by observables which can be systematically refined to approximate observables.

More general properties (such as those used in program verification) can only be handled through assertions in a suitable formal language. We are currently looking into the problem of modeling these properties using abstract interpretation techniques, with the goal of handling them as approximate observables. This would make our results applicable to the more general case.

Abstract diagnosis can be turned into a more practical tool by extending it to the case of real-life logic languages, e.g., PROLOG. It is worth noting that this can easily be done, once we have a semantic framework for PROLOG similar to the one we have used here for pure logic programs. Such a framework has indeed already been defined [71]. It handles almost all the impure features of PROLOG (including cut). The collecting semantics has all the properties which are relevant to diagnosis, namely the condensing property and the equivalence between operational and denotational denotations.

Another relevant issue is related to the abstract operations associated to an observable. In this thesis we have considered the case where the abstract operations

are systematically derived from the concrete ones and are optimal. Our results are still valid if the optimal abstract operations are replaced by (correct) non-optimal operations, as one does in most practical abstract interpretation frameworks. Since this leads to a further loss of precision, the accuracy of the diagnosis method will of course be affected. The same arguments apply to the case of non-condensing observables.

Finally, let us mention that some recent results on the reconstruction of verification techniques by abstract interpretation [72] show that it is quite easy to extend abstract diagnosis to specifications consisting of pre- and post-conditions. Handling pre-conditions simply requires the choice of a more concrete collecting semantics, modeling call patterns in addition to computed answers.

5.8 The Implementation of the Observable Modules

5.8.1 The Computed Answer Observable (Id)

```
observableAnswers(subst, A, E) :-
    copy_term(A, A1), oracle(subst, A1, E).

observableClause(subst, A, E, B, C) :-
    copy_term(A,A2), clause(A2,B2), freecopy(B2,B),
    E = (A2:-B2), copy_term(E,C).

domainBottom(subst, emptyset).

domainMeet(subst, E, E1, (A1 :- Bs)) :-
    copy_term(E,(A1:-(B1,Bs))), B1 = E1.
domainMeet(subst, E, E1, (A1 :- true)) :-
    copy_term(E,(A1:-B1)), B1 = E1.

domainProject(subst, _A, (A1:- _), A1).

domainJoin(subst, emptyset,Xs,Xs) :- !.
domainJoin(subst, Xs,emptyset,Xs) :- !.
domainJoin(subst, Xs,Ys,Xs).
domainJoin(subst, Xs,Ys,Ys).
```

```
domainSingleton(subst,E,Es) :- var(E), E=Es.
domainSingleton(subst,E,Es) :- nonvar(E), variant(E,Es).

readObs(subst,A,E) :- read(E).
showAelement(subst,A,E,E).

freecopy((A,As),(B,Bs)) :-
    !, freecopy(As,Bs), functor(A,F,N), functor(B,F,N).
freecopy(A,B) :- !,functor(A,F,N), functor(B,F,N).

variant(T1,T2) :-
    copy_term(T2,T3),
    not not ( (numbervars(T1,0,N), numbervars(T3,0,N), T1 = T3)).
```

5.8.2 The *depth*(k) Observable (τ_k)

```
observableAnswers(depth(K), A, E) :-
    copy_term(A, A1), oracle(depth(K), A1, E).

observableClause(depth(K), A, E, B, C) :-
    observableClause(subst, A, E1, B, C),
    K1 is K+2, depth(K1, E1, E).

domainBottom(depth(_K), Bottom) :- domainBottom(subst, Bottom).

domainProject(depth(_K), A, E, Esub) :-
    domainProject(subst, A, E, Esub).

domainMeet(depth(K), E, E1, EdepK) :-
    domainMeet(subst, E, E1, Esub),
    K1 is K+2, depth(K1, Esub, EdepK).

domainJoin(depth(_K), E, E1s, E2s) :- domainJoin(subst,E,E1s,E2s).

domainSingleton(depth(_K), E, E1) :- domainSingleton(subst,E,E1).

readObs(depth(K),A,EdepK) :- readObs(subst,A,Esub),K1 is K+1,
    depth(K1, Esub, EdepK).
showAelement(depth(K),A,E,Ae) :- showAelement(subst,A,E,Ae).
```

```
depth(0, _T, X):-!.
depth(K, T, T1) :- K>0, compound(T), !, functor(T,F,N),
    functor(T1,F,N), K1 is K-1, depth(N, K1, T, T1).
depth(K, T, T).

depth(0, _K, _, _).
depth(N, K, T, T1) :-
    N >0, arg(N, T, A), depth(K, A, A1), arg(N, T1, A1),
    N1 is N-1, depth(N1, K, T, T1).
```

5.8.3 The $\mathcal{POS}$ Observable (Υ)

The implementation of $\mathcal{POS}$ is clearly much more complex than the previous ones. The following programs are essentially an interface plus the $\mathcal{POS}$ implementation adapted from an original abstract interpretation framework for PROLOG [93].

```
observableAnswers(pos(subst), A, Epos) :-
    oracle(pos(subst), A, F), posubToPos(F,Epos).

observableClause(pos(subst), A, Epos, B, C) :-
    observableClause(subst, A, E, B, C), posubEqua((A:-B), E, F),
    posubToPos(F,Ebig), varsOf((A,B),Vars),
    posProj(Ebig, Vars, Epos).

domainBottom(pos(subst), X) :- posBottom(X).

domainJoin(pos(subst), F1, F2, F) :- posJoin(F1, F2, F).

domainMeet(pos(subst), F1, F2, F) :- posMeet(F1, F2, F).

domainProject(pos(subst), A, F, Fr) :-
    varsOf(A,Vars), posProj(F, Vars, Fr).

domainSingleton(pos(subst), F, Fs) :- posImplicant(F, Fs).

readObs(pos(subst),A,E) :- read(X), (X \== no -> X=(A,E); X=E),
    assert( (oracleMemory(Obs, A, no_more_answers)) ).

showAelement(pos(subst),A,or([E]),(A,E)):-!.
```

```
showAelement(pos(subst),A,E,(A,E)).

varsOf(T,Xs) :- varsof(T,Ys),removeReps([],_SeenOut,Ys,Xs).
varsof(X,[X]) :- var(X),!.
varsof(A,Vars) :- A =.. [_|L], varsof(L,[],Vars).
varsof([],L,L).
varsof([A|As],L,Vs) :- varsof(A,AL),append(AL,L,G),
    varsof(As,G,Vs).

posubEqua((A:-(B,C)),(EA:-(EB,EC)),and(F1,F2)) :-
    !,posubEqua((A:-B),(EA:-EB),F1),posubEqua(C,EC,F2).
posubEqua((A:-B),(EA:-EB),and(F1,F2)) :-
    !,posubEqua(A,EA,F1),posubEqua(B,EB,F2).
posubEqua(true,true,true):-!.
posubEqua(A, E, F):-
    functor(A,Fun,N),functor(E,Fun,N), posubEqua(N, A, E, F).

posubEqua(1, A, E, Fiff) :- arg(1,A,V), arg(1,E,T),
    posubEquaIff(V, T, Fiff).
posubEqua(N, A, E, and(Fiff,F2)) :-
    N >1, arg(N,A,V), arg(N,E,T), posubEquaIff(V, T, Fiff),
    N1 is N-1, posubEqua(N1, A, E, F2).

posubEquaIff(V, T, V) :- posubTerm(T, true),!.
posubEquaIff(V, T, iff(V,F)) :- posubTerm(T, F).

posubTerm(T,true) :- atomic(T),!.
posubTerm(T,T) :- var(T),!.
posubTerm(T, F) :- compound(T),functor(T,_Name,N),
    posubTerm(N, T, F).

posubTerm(1, T, F):- arg(1, T, A), posubTerm(A, F).
posubTerm(N, T, F) :- N >1, arg(N, T, A), posubTerm(A, true), !,
    N1 is N-1, posubTerm(N1, T, F).
posubTerm(N, T, F) :- N >1, N1 is N-1, posubTerm(N1, T, true), !,
    arg(N, T, A), posubTerm(A, F).
posubTerm(N, T, and(F1,F2)) :- N >1, arg(N, T, A),
    posubTerm(A, F1), N1 is N-1, posubTerm(N1, T, F2).
```

```
posubToPos(X,F):- var(X),!,posVar(X,F).
posubToPos(true,X) :- posTop(X).
posubToPos(false,X) :- posBottom(X).
posubToPos(and(F1,F2),V) :-
    posubToPos(F1,V1), posubToPos(F2,V2), posMeet(V1,V2,V).
posubToPos(or(F1,F2),V) :-
    posubToPos(F1,V1), posubToPos(F2,V2), posJoin(V1,V2,V).
posubToPos(iff(Var,F),R) :- posubToPos(F,V), posIff(Var,V,R).

/* POS domain */

posTop(or([and([])])).

posBottom(or([])).

posVar(X,or([and([X])])).

/* posIff */

posIff(Var,or([and(T)]),or([ and([Var|T]) |Cs])) :-
    onlyIf(T,Var,or(Cs)).

onlyIf([],_Var,or([])).
onlyIf([Hneg|T],Var,or([ and([-Var,H]) |Cs])) :-
    nonvar(Hneg), Hneg = -H, !, onlyIf(T,Var,or(Cs)).
onlyIf([H|T],Var,or([ and([-Var,-H]) |Cs])) :- onlyIf(T,Var,or(Cs)).

/* posJoin */

posJoin(or(L1),or(L2),C) :- append(L1,L2,L), simplify2(or(L),C1),
    simplify3(C1,C2),simplify2(C2,C).

simplify2(C,CS) :- powersOfTwo(P), compNums(C,P,Nums,[],_),
    selectEntailed([],1,Nums,Ent), sort(Ent,OrdEnt),
    removeEntailed(1,C,OrdEnt,CS).

powersOfTwo([1,2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,
32768,65536,131072,262144,524288,1048576,2097152,4194304,8388608,
16777216,33554432,67108864,134217728]).
```

```
compNums(or([]),_Table,[],_Seen, []).
compNums(or([H|T]),Table,[NH|NT],Seen,Lout) :-
    compNums(H,NH,Table,TableOut,Seen,SeenOut,Lin,Lout),
    compNums(or(T),TableOut,NT,SeenOut,Lin).

compNums(and([]),0,Table,Table,Seen,Seen,Lin,Lin).
compNums(and([X]),Num,Table,Table,Seen,Seen,Lin,Lin) :-
    funnyMember((X,Num),Seen),!.
compNums(and([X]),Num,[Num|TT],TT,Seen,[(X,Num)|Seen],Lin,Lout) :-
    !, update(X,Num,Seen,Lin,Lout).
compNums(and([X|T]),Num,Table,TableOut,Seen,SeenOut,Lin,Lout) :-
    funnyMember((X,Num1),Seen), !,
    compNums(and(T),Num2,Table,TableOut,Seen,SeenOut,Lin,Lout),
    Num is Num1+Num2.
compNums(and([X|T]),Num,[Num1|TT],TableOut,Seen,SeenOut,Lin,Lout) :-
    update(X,Num1,Seen,Lin,L1),
    compNums(and(T),Num2,TT,TableOut,[(X,Num1)|Seen],SeenOut,L1,
    Lout),
    Num is Num1+Num2.

update(X, Num1, Seen, L, [(X,Num1,Num2)|L]) :- var(X),
    funnyMember((-X,Num2),Seen),!.
update(-X, Num1, Seen, L, [(X,Num1,Num2)|L]) :- var(X),
    funnyMember((X,Num2),Seen),!.
update(_, _, _, L, L).

funnyMember((Var,Num),[(Var1,Num)|_T]) :- Var==Var1,!.
funnyMember(X,[_H|T]) :- funnyMember(X,T).

selectEntailed(_Seen,_K,[],[]).
selectEntailed(Seen,K,[H|T],[K|T1]) :-
    isEntailed(H,Seen), !, K1 is K+1, selectEntailed(Seen,K1,T,T1).
selectEntailed(Seen,K,[H|T],Result) :- entailed(H,Seen,E), K1 is K+1,
    append(E,T1,Result), selectEntailed([(H,K)|Seen],K1,T,T1).

isEntailed(N,[(M,_Pos)|_T]) :- entails(M,N), !.
isEntailed(N,[_H|T]) :- isEntailed(N,T).
```

```
entailed(_N,[],[]).
entailed(N,[(M,Pos)|T],[Pos|T1]) :- entails(N,M), !,
    entailed(N,T,T1).
entailed(N,[_H|T],T1) :- entailed(N,T,T1).

entails(N,NEntailed) :- Tmp is N /\ NEntailed,Tmp=N.

removeEntailed(_K,or([]),[],or([])).
removeEntailed(K,or([_H|T]),[K|TE],or(T1)) :-
    !, K1 is K+1, removeEntailed(K1,or(T),TE,or(T1)).
removeEntailed(K,or([H|T]),Entailed,or([H|T1])) :-
    K1 is K+1, removeEntailed(K1,or(T),Entailed,or(T1)).

simplify3(or(L),or(LS)) :- powersOfTwo(P),
    compNums(or(L),P,Nums,[],Sus), joinNumsC(Sus,Nums,L,_,LS).

joinNumsC([],Ns,As,Ns,As).
joinNumsC([_|_],[],[],[],[]).
joinNumsC([(X,P1,P2)|Ds],[N|Ns],[A|As],N3s,A3s) :-
    joinNumsC(X,P1,P2,N,Ns,A,As,N2,N1s,A2,A1s),
    joinNumsC([(X,P1,P2)|Ds],N1s,A1s,N2s,A2s),
    joinNumsC(Ds,[N2|N2s],[A2|A2s],N3s,A3s).

joinNumsC(_,_,_,N,[],A,[],N,[],A,[]).
joinNumsC(X,P1,P2,N1,[N2|Ns],and(L1),[_A2|As],N3,Ns,and(L2),As) :-
    canjoin(N1,N2,P1,P2), !, N3 is (N1 /\ \(P1)) /\ \(P2),
    softDelete(X, L1, L2).
joinNumsC(X,P1,P2,N1,[N|Ns],A1,[A|As],N2,[N|N2s],A2,[A|A2s]) :-
    joinNumsC(X,P1,P2,N1,Ns,A1,As,N2,N2s,A2,A2s).

canjoin(N1,N2,P1,P2) :- P is P1+P2,
    N is (N1 /\ \(N2)) \/ (N2 /\ \(N1)), P=N,
    (X is N1 /\ P, X=P1, Y is N2 /\ P, Y=P2 ;
    X is N1 /\ P, X=P2, Y is N2 /\ P, Y=P1).

softDelete(X, [Y|Ys], Ys) :- X==Y, !.
softDelete(X, [Y|Ys], [Y|Zs]) :- softDelete(X,Ys,Zs).

/* posMeet */
```

```
posMeet(or([]),_,or([])).
posMeet(or([H|T]),L,Result) :- product(H,L,or(H1)),
    append(H1,T1,Res), posMeet(or(T),L,or(T1)),
    simplify1(or(Res),Resu), simplify2(Resu,Resul),
    simplify3(Resul,Result).

product(and(_L),or([]),or([])).
product(and(L),or([and(H)|T]),or([and(H1)|T1])) :-
    append(L,H,H1), product(and(L),or(T),or(T1)).

simplify1(or([]),or([])).
simplify1(or([H|T]),or(Result)):-simplify1Aux([],H,H1,Flag),
     (Flag=yes -> Result=T1 ;Result=[H1|T1]),
     simplify1(or(T),or(T1)).

simplify1Aux(_Seen,and([]),and([]),no).
simplify1Aux(Seen,and([H|T]),and(T1),Flag) :- memberVar(H,Seen,How),
    (How=pos -> simplify1Aux([H|Seen],and(T),and(T1),Flag) ;
    (How=neg -> Flag=yes ;
    (simplify1Aux([H|Seen],and(T),and(T2),Flag),T1=[H|T2]))).

memberVar(_Var,[],no).
memberVar(Var,[H|_T],pos) :- Var==H,!.
memberVar(-Var,[H|_T],neg) :- Var==H,!.
memberVar(Var,[-H|_T],neg) :- Var==H,!.
memberVar(Var,[_H|T],How) :- memberVar(Var,T,How).

/* posProj */

posProj(Const,Vars,Result) :- constVars(Const,LV),
    diff(LV,Vars,Diff), posRestrict(Const,Diff,Res),
    simplify3(Res,Result).

constVars(or(List),Xs):-
    constVarsWRep(List,Ys),removeReps([],_SeenOut,Ys,Xs).

constVarsWRep([],[]).
constVarsWRep([and(C)|Cs],Rs) :- constVarsWRep(Cs,Ds),
```

```
    constVarsWRep(C,Ds,Rs).

constVarsWRep([],Ds,Ds).
constVarsWRep([Hneg|Cs],Ds,[H|Rs]) :-
    nonvar(Hneg), Hneg = -H, !, constVarsWRep(Cs,Ds,Rs).
constVarsWRep([H|Cs],Ds,[H|Rs]) :- constVarsWRep(Cs,Ds,Rs).

removeReps(Seen,Seen,[],[]).
removeReps(SeenIn,SeenOut,[H|T],T1) :-
    softMember(H,SeenIn),!, removeReps(SeenIn,SeenOut,T,T1).
removeReps(SeenIn,SeenOut,[H|T],[H|T1]) :-
    removeReps([H|SeenIn],SeenOut,T,T1).

softMember(X,[Y|_L]) :- X==Y,!.
softMember(X,[_Y|L]) :- softMember(X,L).

diff([],_L,[]).
diff([H|T],L,T1) :- softMember(H,L), !, diff(T,L,T1).
diff([H|T],L,[H|T1]) :- diff(T,L,T1).

posRestrict(C,[],C).
posRestrict(C,[H|T],CR):-
    schroder(C,H,CS1), simplify1(CS1,CS2), simplify2(CS2,CS),
    posRestrict(CS,T,CR).

schroder(or([]),_V,or([])).
schroder(or([ and(A) | O]),V,or([ and(AwoV) | OwoV])) :-
    kill(V,A,AwoV), schroder(or(O),V,or(OwoV)).

kill(_V,[],[]).
kill(V,[X|Xs],Xs) :- X == V.
kill(V,[X|Xs],Xs) :- X == -V.
kill(V,[X|Xs],[X|Ys]) :- X \== V, X \== -V, kill(V,Xs,Ys).

/* posImplicant */

posImplicant(or([F]),or(Fs)) :- var(F), member(F,Fs).
posImplicant(or([F]),or(Fs)) :- nonvar(F), powersOfTwo(P),
    compNums(or([F|Fs]),P,[N|Ns],[],_), memberEntails(N,Ns).
```

```
memberEntails(V,[N|_Ns]) :- entails(N,V),!.
memberEntails(V,[_N|Ns]) :- memberEntails(V,Ns).
```

Part III

Epilogue

Concluding Remarks

In Part I we introduced a uniform framework for the *reconstruction* of existing semantics and for the *systematic* design of *new* semantics able to deal also with the *approximation* typical of static *program analysis*. Our technical tools are that of abstract interpretation. The idea of using abstract interpretation techniques as unifying framework for various semantics is well-known [36]. However the originality of this thesis is that we fully exploit this idea and provide a real flexible framework which provides useful theoretical bases for *new* semantic-based applications.

Our results allow us to derive in a systematic way several known semantics, together with their properties. Perfect observables allow us to reconstruct the resultants semantics in [54] and the Heyting semantics in [67]. The results on denotational observables apply to the least Herbrand model, the atomic logical consequence semantics [16], the s-semantics [45, 13], the partial answer semantics [43] and the call pattern semantics [55]. The results on semi-denotational observables apply to bottom-up abstract interpretation frameworks, such as those whose collecting semantics is the computed answer semantics [9, 17], the correct answer semantics [59] and the call pattern semantics [51]. Finally, the results on semi-perfect observables apply to top-down abstract interpretation frameworks [15].

Furthermore, our taxonomy of observables can be useful to design a *new* semantics with some *a priori* given properties. One need only to define an observable and check if it belongs to the right class, since the axioms for a class are sufficient conditions for the derived semantics to have the required properties. The advantage of this approach is that we can choose the appropriate level of abstraction for specializing the semantics operators of the framework.

We discussed how our results can be useful in practical applications and showed how our results give some new insights into some classical controversial issues, such as top-down analysis versus bottom-up analysis and *goal-dependence* versus *goal-independence*.

An application which is currently under study is the *systematic* design of observables with specific semantic properties. The problem can be stated as follows. We want to model the observable α, by a semantics which has some property (such

as being precise, *AND*-compositional, *OR*-compositional, goal-independent, or top-down). If α belongs to a class which does not enjoy that property, we need to determine another observable β which is more concrete than α (and therefore is correct w.r.t. it) and which has the required property.

Consider, for example, the case where one wants to model computed answers by an *OR*-compositional semantics (because needs to reason in a modular way). Computed answers are denotational and not perfect and therefore they are not *OR*-compositional. An *OR*-compositional semantics, correct w.r.t. computed answers, was defined in [14]. It turns out to be exactly the semantics for the observable computed resultants, which is indeed perfect and therefore *OR*-compositional.

As another example, assume one wants to model groundness dependencies of computed answers by an *AND*-compositional (goal-independent) semantics. The simple "groundness" observable is not even semi-denotational. On the other hand, $\mathcal{POS}$ is correct w.r.t. groundness dependencies of computed answers and is semi-denotational (and therefore *AND*-compositional).

The theory of abstract interpretation provides tools for the systematic construction of "more concrete" observables, i.e., *refinement operators*. Examples of refinement operators are reduced product, disjunctive completion, functional dependencies and Heyting completion. The above mentioned examples can be handled by these techniques. In particular [60] shows that the resultants semantics can be obtained by refining the domain of the s-semantics by functional dependencies, while [62, 89] reconstruct $\mathcal{POS}$ from the "groundness" observable by Heyting completion. Within our framework, one can handle the problem of establishing general results about the properties of a class of observables and those of their refinements. Up to now, if the observable does not belong to the right class, by focusing on the axioms which are not satisfied, the "observable designer" can find useful suggestions on how to transform an unsatisfactory abstraction into a good observable, by using this refinement techniques. However, we believe that the framework can be extended, by using refinement techniques, to systematically derive the observable with the desired properties from one which does not enjoy them.

In Part II we have introduced several effective techniques to perform the diagnosis of pure logic programs.

Partial diagnosis can be used whenever we have a (finite) partial knowledge about the intended behavior. This knowledge can be derived from symptoms detection (performed using testing techniques) and symptom-directed queries to the user, as in the practical symptom-directed debuggers. Our results justifies the process of determining incorrect clauses from incorrectness symptoms and tells us that incorrect clauses do always correspond to errors. Symptom-directed debuggers are of course not concerned with the problem of deciding partial correctness. On the other

side, symptom-directed debuggers derive p-uncovered $\mathbb{A}$-elements starting from an incompleteness symptom. Our negative result on p-uncovered $\mathbb{A}$-elements shows that the uncovered $\mathbb{A}$-element does not always correspond to an incompleteness error, unless the oracle can return infinite answers. One might also think of other partial knowledge acquisition techniques.

Diagnosis w.r.t. approximate observables is instead useful when one wants to perform the diagnosis w.r.t. properties which can be modeled by abstractions over Noetherian domains. This is interesting because several useful properties are modeled by approximate observables or by observables which can be systematically refined to approximate observables. The results are useful and comparable to those obtained by verification techniques. In fact, if we consider the case where specifications consist of post-conditions only, both abstract diagnosis and verification provide *a sufficient condition for partial correctness* and for both techniques there is no sufficient condition for completeness. In order to verify completeness, we have to rely on a fixpoint which, in general, cannot be computed in a finite number of steps. As expected, abstract diagnosis (whose aim is locating bugs rather than just proving correctness) gives us also information useful for debugging, by means of incorrect clauses and uncovered elements.

As already mentioned, the approximation given by approximate observables is somewhat dual of the one given by partial diagnosis. This is reflected by the duality of the results. In fact, in partial diagnosis we have a sufficient condition for completeness instead of a sufficient condition for partial correctness. Moreover, p-incorrect clauses always correspond to bugs, while this is not the case for p-uncovered elements. This suggests that we might improve our results on completeness by taking an under-approximation as abstraction. Of course, if we want to preserve the results on partial correctness, we need to keep also the standard over-approximation. This would lead us to a theory of abstract diagnosis, where two different abstractions are used at the same time. This would in turn require a specification consisting of a pair of abstractions of the intended concrete semantics. This is an interesting idea for further research that might build upon the already mentioned approach by Ferrand, which works with pairs of specifications.

More general properties (such as those used in program verification) can only be handled through assertions in a suitable formal language. We are currently looking into the problem of modeling these properties using abstract interpretation techniques, with the goal of handling them as approximate observables. This would make our results applicable to the more general case.

Let us note that Abstract Diagnosis can be turned into a more practical tool by extending it to the case of real-life logic languages, e.g., PROLOG. It is worth noting that this can easily be done, once we have a semantic framework for PROLOG

similar to the one we have used here for pure logic programs. Such a framework has indeed already been defined [71]. It handles almost all the impure features of PROLOG (including cut). The collecting semantics has all the properties which are relevant to diagnosis, namely the condensing property and the equivalence between operational and denotational semantics.

Another relevant issue is related to the abstract operations associated to an observable. In this thesis we have considered the case where the abstract operations are systematically derived from the concrete ones and are optimal. Our results are still valid if the optimal abstract operations are replaced by (correct) non-optimal operations, as one does in most practical abstract interpretation frameworks. Since this leads to a further loss of precision, the accuracy of the diagnosis method will of course be affected. The same arguments apply to the case of non-condensing observables.

Let us simply mention that some recent results on the reconstruction of verification techniques by abstract interpretation [72] show that it is quite easy to extend abstract diagnosis to specifications consisting of pre- and post-conditions. Handling pre-conditions simply requires the choice of a more concrete collecting semantics, modeling call patterns in addition to computed answers.

As a final remark, we want to point out that, since it is based on standard operational and denotational semantics definitions, our approach can be generalized to other programming languages. We just need to define a concrete denotational and operational semantics on the same concrete semantic domain. The compositionality properties will be of course different and related to the language syntactic operators. Looking at the axioms of observables in Chapter 3 and the ones in Chapter 4 one would expect a kind of general operator-independent definition capturing the phenomenon, since the nature of both sets of axioms is similar, the difference is only in the semantic operators used. Going further in that direction could we give an operator-independent characterization for other classes of observables. We believe that our semantic framework will allow us to look at the design of the abstract operators as a process where, in most cases, we just choose among the best approximations of semantic operations ad different levels of definition. The choice will eventually be made as usual as a trade-off between efficiency and precision.

Bibliography

[1] S. Abramsky and C. Hankin. An Introduction to Abstract Interpretation. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*, pages 9–31. Ellis Horwood Ltd, 1987. 1.4

[2] G. Amato and G. Levi. Properties of the lattice of observables in logic programming. In M. Falaschi and M. Navarro, editors, *Proceedings of the APPIA-GULP-PRODE'97 Joint Conference on Declarative Programming*, 1997. 3.7, 3.8

[3] K. R. Apt. Introduction to Logic Programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 495–574. Elsevier and The MIT Press, 1990. 1, 1.3, 3, 1.3.3, 3.3.2

[4] K. R. Apt. *From Logic Programming to Prolog*. Prentice Hall, 1997. 5.1.1, 5.5.1

[5] K. R. Apt and E. Marchiori. Reasoning about Prolog programs: from modes through types to assertions. *Formal Aspects of Computing*, 6(6A):743–765, 1994. 5.1.1, 5.5.1

[6] K. R. Apt and D. Pedreschi. Reasoning about termination of pure PROLOG programs. *Information and Computation*, 106(1):109–157, 1993. 5.3.1, 5.3.7, 5.3.1

[7] K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, 1982. 1.3.3, 1.3.3, 1.3.9, 1.3.3, 1.3.11

[8] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Boolean functions for dependency analysis: Algebraic properties and efficient representation. In B. Le Charlier, editor, *Proc. Static Analysis Symposium, SAS'94*, volume 864 of *Lecture Notes in Computer Science*, pages 266–280. Springer-Verlag, 1994. 3.5.1, 3.7, 32, 4.4.1, 5.1.3

[9] R. Barbuti, R. Giacobazzi, and G. Levi. A General Framework for Semantics-based Bottom-up Abstract Interpretation of Logic Programs. *ACM Transactions on Programming Languages and Systems*, 15(1):133–181, 1993. 3.7, III

[10] G. Birkhoff. Lattice Theory. In *AMS Colloquium Publication, third ed.*, 1967. 1, 1.4.1

[11] G. Birkhoff and S. MacLane. *A Survey of Modern Algebra*, volume 25. MacMillan, 1965. Third edition. 1

[12] A. Bossi and N. Cocco. Verifying correctness of logic programs. In J. Diaz and F. Orejas, editors, *Proc. TAPSOFT'89*, pages 96–110, 1989. 5.1.1

[13] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *Journal of Logic Programming*, 19–20:149–197, 1994. I.4, 1.3.3, 1.3.4, 4, 2, 2.1, 2.6, 3.4.1, 3.7, III

[14] A. Bossi, M. Gabbrielli, G. Levi, and M. C. Meo. A Compositional Semantics for Logic Programs. *Theoretical Computer Science*, 122(1–2):3–47, 1994. 3, 3.8, 5.6, III

[15] M. Bruynooghe. A Practical Framework for the Abstract Interpretation of Logic Programs. *Journal of Logic Programming*, 10(2):91–124, 1991. Revised version of K.U.L. technical report CW–62, 1987. I.4, 3, 3.6, 3.7, 4, III

[16] K. L. Clark. Predicate logic as a computational formalism. Res. Report DOC 79/59, Imperial College, Dept. of Computing, London, 1979. 1.3.10, 1.3.12, 1.3.14, 4, 2.4.1, 3, 3.4.2, 3.7, 4.3.1, III

[17] M. Codish, D. Dams, and E. Yardeni. Bottom-up Abstract Interpretation of Logic Programs. *Theoretical Computer Science*, 124(1):93–125, 1994. 3.7, III

[18] M. Codish, S. K. Debray, and R. Giacobazzi. Compositional Analysis of Modular Logic Programs. In *Proc. Twentieth Annual ACM Symp. on Principles of Programming Languages*, pages 451–464. ACM Press, 1993. 5.6

[19] M. Comini. Sources of the abstract diagnosis meta-interpreters. Available at http://www.di.unipi.it/~comini/Projects/. 4

[20] M. Comini and G. Levi. An algebraic theory of observables. In M. Bruynooghe, editor, *Proceedings of the 1994 Int'l Symposium on Logic Programming*, pages 172–186. The MIT Press, 1994. (document), 3

[21] M. Comini, G. Levi, and M. C. Meo. Compositionality of *SLD*-derivations and their abstractions. In J. Lloyd, editor, *Proceedings of the 1995 Int'l Symposium on Logic Programming*, pages 561–575. The MIT Press, 1995. 2, 3

[22] M. Comini, G. Levi, and M. C. Meo. A theory of observables for logic programs. Submitted for publication. http://www.di.unipi.it/~comini/Papers/, 1996. 2, 3

[23] M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Abstract diagnosis. Submitted for publication. http://www.di.unipi.it/~comini/Papers/, 1996. 3.8, 4, 5.1.4

[24] M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Proving properties of logic programs by abstract diagnosis. In M. Dams, editor, *Analysis and Verification of Multiple-Agent Languages, 5th LOMAPS Workshop*, volume 1192 of *Lecture Notes in Computer Science*, pages 22–50. Springer-Verlag, 1996. 3.8, 5.1.4

[25] M. Comini, G. Levi, and G. Vitiello. Abstract debugging of logic programs. In L. Fribourg and F. Turini, editors, *Proc. Logic Program Synthesis and Transformation and Metaprogramming in Logic 1994*, volume 883 of *Lecture Notes in Computer Science*, pages 440–450. Springer-Verlag, 1994. 4, 4.5, 5.1.3, 5.1.4

[26] M. Comini, G. Levi, and G. Vitiello. Declarative diagnosis revisited. In J. Lloyd, editor, *Proceedings of the 1995 Int'l Symposium on Logic Programming*, pages 275–287. The MIT Press, 1995. 3.8, 5.1.4, 5.3

[27] M. Comini, G. Levi, and G. Vitiello. Efficient detection of incompleteness errors in the abstract debugging of logic programs. In M. Ducassé, editor, *Proc. 2nd International Workshop on Automated and Algorithmic Debugging, AADEBUG'95*, 1995. 4, 4.5, 5.1.3, 5.1.4

[28] M. Comini, G. Levi, and G. Vitiello. On the Abstract Diagnosis of Logic Programs. In M.I. Sessa, editor, *Proceedings GULP-PRODE'95*, 1995. 5.1.4

[29] M. Comini, G. Levi, and G. Vitiello. Modular abstract diagnosis. In *International Workshop on Tools and Environments for (Constraint) Logic Programming, ILPS'97 Postconference Workshop*, 1997. 5.1.4

[30] M. Comini and M. C. Meo. Compositionality properties of *SLD*-derivations. *Theoretical Computer Science*, 1997. To appear. 2

[31] A. Cortesi, G. Filè, and W. Winsborough. *Prop* revisited: Propositional Formula as Abstract Domain for Groundness Analysis. In *Proc. Sixth IEEE Symp.*

on Logic In Computer Science, pages 322–327. IEEE Computer Society Press, 1991. 3.5.1, 4.4.1

[32] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. Fourth ACM Symp. Principles of Programming Languages*, pages 238–252, 1977. I.2, 1.4, 2, 3, 4.2, 5.1.2

[33] P. Cousot and R. Cousot. A constructive characterization of the lattices of all retracts, pre-closure, quasi-closure and closure operators on a complete lattice. *Portugaliæ Mathematica*, 38(2):185–198, 1979. 1.4.1

[34] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proc. Sixth ACM Symp. Principles of Programming Languages*, pages 269–282, 1979. I.2, 1.4, 1.4.1, 1.4.2, 3, 3.1, 4.2, 5.1.2

[35] P. Cousot and R. Cousot. Abstract Interpretation and Applications to Logic Programs. *Journal of Logic Programming*, 13(2 & 3):103–179, 1992. 1.4.2

[36] P. Cousot and R. Cousot. Abstract Interpretation Frameworks. *Journal of Logic and Computation*, 2(4):511–549, 1992. (document), III

[37] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages). In *Proceedings of the IEEE International Conference on Computer Languages (ICCL'94)*, pages 95–112. IEEE Computer Society Press, 1994. 5

[38] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997. 5

[39] S. K. Debray and P. Mishra. Denotational and Operational Semantics for Prolog. *Journal of Logic Programming*, 5:61–91, 1988. 2.2

[40] W. Drabent and J. Maluszynski. Inductive assertion method for logic programs. *Theoretical Computer Science*, 59(1):133–155, 1988. 5.1.1

[41] W. Drabent, S. Nadjm-Tehrani, and J. Maluszynski. Algorithmic debugging with assertions. In H. Abramson and M.H. Rogers, editors, *Meta-Programming in Logic Programming*, pages 383–398. MIT Press, 1989. 5.1.1, 5.3.2

[42] E. Eder. Properties of substitutions and unification. *Journal of Symbolic Computation*, 1:31–46, 1985. 1.3.2, 1.3.2, 1.3.2

[43] M. Falaschi and G. Levi. Finite failures and partial computations in concurrent logic languages. *Theoretical Computer Science*, 75:45–66, 1990. 3, 3.7, III

[44] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A new Declarative Semantics for Logic Languages. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 993–1005. The MIT Press, 1988. 1.3.16

[45] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318, 1989. I.1, 1.3.4, 1.3.16, 4, 2, 3, 3.4.1, 3.4.2, 3.7, 4.3.1, III

[46] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs. *Information and Computation*, 102(1):86–113, 1993. 1.3.4

[47] G. Ferrand. A reconstruction of Logic Programming with Negation. Technical Report No. 86/5, Lab. d'Informatique, Département de Mathématiques et d'Informatique, Université d'Orléans, 1986. 4, 3.4.2

[48] G. Ferrand. Error Diagnosis in Logic Programming, an Adaptation of E. Y. Shapiro's Method. *Journal of Logic Programming*, 4:177–198, 1987. (document), I.3, I.4, 4, 2.4.1, 3, 5.1.1, 5.1.3, 5.2, 5.2, 5.3, 5.3

[49] G. Ferrand. The notions of symptom and error in declarative diagnosis of logic programs. In P. A. Fritzson, editor, *Automated and Algorithmic Debugging, Proc. AADEBUG '93*, volume 749 of *Lecture Notes in Computer Science*, pages 40–57. Springer-Verlag, 1993. 5.2, 5.4.2, 5.5.1

[50] G. Filè, R. Giacobazzi, and F. Ranzato. A unifying view on abstract domain design. *ACM Computing Surveys*, 28(2):333–336, 1996. I.2, 3.8, 5.1.2

[51] M. Gabbrielli and R. Giacobazzi. Goal independency and call patterns in the analysis of logic programs. In J. Urban E. Deaton, D. Oppenheim and H. Berghel, editors, *Proceedings of the Ninth ACM Symposium on Applied Computing*, pages 394–399, Phoenix AZ, 1994. ACM Press. 3.4.3, 3.7, III

[52] M. Gabbrielli, G. Levi, and M. C. Meo. Observational Equivalences for Logic Programs. In K. Apt, editor, *Proc. Joint Int'l Conf. and Symposium on Logic Programming*, pages 131–145. The MIT Press, 1992. 2, 2.1

[53] M Gabbrielli, G. Levi, and M. C. Meo. Observable Behaviors and Equivalences of Logic Programs. *Information and Computation*, 122(1):1–29, 1995. (document), 3, 3.4.3

[54] M. Gabbrielli, G. Levi, and M. C. Meo. Resultants semantics for PROLOG. *Journal of Logic and Computation*, 6(4):491–521, 1996. 1.3.4, 1.3.3, 1.3.8, 1.3.15, 1.5, 2, 3, 3.3.2, 3.3.1, 3.7, III

[55] M. Gabbrielli and M. C. Meo. Fixpoint Semantics for Partial Computed Answer Substitutions and Call Patterns. In H. Kirchner and G. Levi, editors, *Algebraic and Logic Programming, Proceedings of the Third International Conference*, volume 632 of *Lecture Notes in Computer Science*, pages 84–99. Springer-Verlag, 1992. 3, 3.7, III

[56] H. Gaifman and E. Shapiro. Fully abstract compositional semantics for logic programs. In *Proc. Sixteenth Annual ACM Symp. on Principles of Programming Languages*, pages 134–142. ACM, 1989. 3

[57] R. Giacobazzi. *Semantic Aspects of Logic Program Analysis*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 1992. (document), I.2

[58] R. Giacobazzi. "Optimal" collecting semantics for analysis in a hierarchy of logic program semantics. In C. Puech and R. Reischuk, editors, *Proc. 13th International Symposium on Theoretical Aspects of Computer Science (STACS'96)*, volume 1046 of *Lecture Notes in Computer Science*, pages 503–514. Springer-Verlag, 1996. (document), 3

[59] R. Giacobazzi, S.K. Debray, and G. Levi. Generalized Semantics and Abstract Interpretation for Constraint Logic Programs. *Journal of Logic Programming*, 25(3):191–247, 1995. 3.7, III

[60] R. Giacobazzi and F. Ranzato. Functional dependencies and Moore-set completions of abstract interpretations and semantics. In J. Lloyd, editor, *Proceedings of the 1995 Interantional Symposium on Logic Programming (ILPS'95)*, pages 321–335. The MIT Press, 1995. 3.8, III

[61] R. Giacobazzi and F. Ranzato. Completeness in abstract interpretation: A domain perspective. In M. Johnson, editor, *Proc. of the 6th International Conference on Algebraic Methodology and Software Technology (AMAST'97)*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1997. 5

[62] R. Giacobazzi and F. Scozzari. Intuitionistic implication in abstract interpretation. In H. Glaser, P. Hartel, and H. Kuchen, editors, *Proceedings of Ninth International Symposium on Programming Languages, Implementations, Logics and Programs PLILP'97*, volume 1292 of *Lecture Notes in Computer Science*, pages 175–189. Springer-Verlag, 1997. I.2, 3.8, 5.1.2, III

[63] P. M. Hill. LUSH-Resolution and its Completeness. Technical Report DCL Memo 78, Department of Artificial Intelligence, University of Edinnburgh, 1974. 1.3.3, 1.3.13

[64] N. D. Jones and A. Mycroft. Stepwise Development of Operational and Denotational Semantics for PROLOG. In Sten-Åke Tärnlund, editor, *Proc. Second Int'l Conf. on Logic Programming*, pages 281–288, 1984. 2.2

[65] N. D. Jones and H. Søndergaard. A Semantics-based Framework for the Abstract Interpretation of PROLOG. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*, pages 123–142. Ellis Horwood Ltd, 1987. I.2, 2.2

[66] R. Kemp and G. Ringwood. An Algebraic Framework for the Abstract Interpretation of Logic Programs. In S. K. Debray and M. Hermenegildo, editors, *Proc. North American Conf. on Logic Programming'90*, pages 506–520. The MIT Press, 1990. 2, 3, 3.3.3, 3.3.2

[67] R. Kemp and G. Ringwood. Reynolds base, Clark Models and Heyting semantics of logic programs. *Theoretical Computer Science*, 1992. To appear. 2, 3, 3.3.3, 3.3.2, 3.7, III

[68] J. L. Lassez, M. J. Maher, and K. Marriott. Unification Revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988. 1.3.2, 1.3.2, 1.3.2

[69] B. Le Charlier, Rossi S., and P. Van Hentenryck. An Abstract Interpretation Framework which Accurately Handles PROLOG Search Rule and the Cut. In M. Bruynooghe, editor, *Proceedings of the 1994 Int'l Symposium on Logic Programming*, pages 157–171. The MIT Press, 1994. I.4, 3, 4

[70] G. Levi and D. Ramundo. A formalization of metaprogramming for real. In D. S. Warren, editor, *Proc. Tenth Int'l Conf. on Logic Programming*, pages 354–373. The MIT Press, 1993. 4

[71] G. Levi and F. Spoto. A Denotational Semantics for Prolog. In M. Falaschi, M. Navarro, and A. Policriti, editors, *Proceedings of the APPIA-GULP-PRODE'97 Joint Conference on Declarative Programming*, pages 201–212, 1997. 5.7, III

[72] G. Levi and P. Volpe. A reconstruction of verification techniques by abstract interpretation. In *ILPS'97. Workshop on Verification, Model Checking and Abstract Interpretation*, 1997. 5.3.2, 5.7, III

[73] Y. Lichtenstein and E. Y. Shapiro. Abstract algorithmic debugging. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. and Symp. on Logic Programming*, pages 512–531, Seattle, 1988. 5.1.1

[74] J. W. Lloyd. Declarative error diagnosis. *New Generation Computing*, 5(2):133–154, 1987. (document), I.3, I.4, 5.1.1, 5.2

[75] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987. Second edition. 1, 1.3, 1.3.3

[76] P. Mancarella and D. Pedreschi. An Algebra of Logic Programs. In R. A. Kowalski and K. A. Bowen, editors, *Proc. Fifth Int'l Conf. on Logic Programming*, pages 1006–1023. The MIT Press, 1988. 5.6

[77] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, NewYork, 1974. 1

[78] K. Marriott and H. Sondergaard. Abstract Interpretation of Logic Programs: the Denotational Approach. In A. Bossi, editor, *Proc. Fifth Italian Conference on Logic Programming*, pages 399–425, 1990. I.2

[79] K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM Letters on Programming Languages and Systems*, 2(1–4):181–196, 1993. 3.5.1, 4.4.1

[80] A. Mycroft. Completeness and predicate-based abstract interpretation. In *Proceedings of the ACM Symp. on Partial Evaluation and Program Manipulation (PEPM '93)*, pages 179–185. ACM Press, New York, 1993. 5

[81] L. Naish. Declarative diagnosis of missing answers. *New Generation Computing*, 10:255–285, 1991. 5.2.2, 36

[82] F. Nielson. A denotational framework for data flow analysis. *Acta Informatica*, 18:265–287, 1982. 1.4

[83] Oystein Ore. Galois Connections. In *Trans. AMS*, volume 55, pages 493–513, 1944. 1.4.2, 1.4.2

[84] C. Palamidessi. Algebraic properties of idempotent substitutions. In M. S. Paterson, editor, *Proc. of the 17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 386–399. Springer-Verlag, 1990. 1.3.2

[85] L. M. Pereira. Rational debugging in logic programming. In E. Y. Shapiro, editor, *Proceedings of the 3rd International Conference on Logic Programming*, volume 225 of *Lecture Notes in Computer Science*, pages 203–210. Springer-Verlag, 1986. I.3, 5.1.3

[86] U. S. Reddy and S. N. Kamin. On the power of abstract interpretation. In *Proc. of the 1992 IEEE Internat. Conf. on Computer Languages* (*ICCL '92*). IEEE Computer Society Press, Los Alamitos, Calif., 1992. 5

[87] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965. 1.3.2, 1.3.1, 1.3.3

[88] T. Sato and H. Tamaki. Enumeration of Success Patterns in Logic Programs. *Theoretical Computer Science*, 34:227–240, 1984. 5.1.3

[89] F. Scozzari. Logical optimality of groundness analysis. In P. Van Hentenryck, editor, *Proceedings of International Static Analysis Symposium, SAS'97*, volume 1302 of *Lecture Notes in Computer Science*, pages 83–97. Springer-Verlag, 1997. I.2, 3.8, 4, 4.5, 5.1.2, 5.1.3, III

[90] R. C. Sekar, P. Mishra, and I. V. Ramakrishnan. On the power and limitation of strictness analysis. *J. ACM*, 44(3):505–525, 1997. 5

[91] E. Y. Shapiro. Algorithmic program debugging. In *Proc. Ninth Annual ACM Symp. on Principles of Programming Languages*, pages 412–531. ACM Press, 1982. (document), I.3, I.4, 5.1.1, 5.1.3, 5.2, 5.2.2, 5.3

[92] E. Y. Shapiro. *Algorithmic Program Debugging*. The MIT Press, 1983. I.4

[93] F. Spoto. Un interprete astratto per programmi logici con cut. In Italian. http://tiramisu.di.unipi.it/˜spoto/analyzer95.ps.gz, December 1995. 5.8.3

[94] L. Sterling and E. Y. Shapiro. *The Art of Prolog*. The MIT Press, 1986. 5.3.1

[95] H. Tamaki and T. Sato. Unfold/Fold Transformations of Logic Programs. In Sten-Åke Tärnlund, editor, *Proc. Second Int'l Conf. on Logic Programming*, pages 127–139, 1984. 3.5.2

[96] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5:285–309, 1955. 1.2.1

[97] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976. 1.3.3, 4

[98] D. A. Wolfram, M. J. Maher, and J. L. Lassez. A Unified Treatment of Resolution Strategies for Logic Programs. In Sten-Åke Tärnlund, editor, *Proc. Second Int'l Conf. on Logic Programming*, pages 263–276, 1984. 1.3.3

Index

Symbols

$\mathsf{A} \cdot \mathsf{D}$ 48
$\mathsf{A} \tilde{\cdot} \mathsf{X}$ 95
$\mathsf{C} \equiv_{\mathbb{C}} \mathsf{C}'$ 168
$\mathsf{D} \underset{\mathsf{P}}{\longmapsto} \mathsf{D}'$ 54
$\mathsf{D} \equiv_{\mathbb{C}} \mathsf{D}'$ 41
$\mathsf{D} \underset{\mathsf{P}}{\longmapsto}^{*} \mathsf{D}'$ 56
$\mathsf{D}\tilde{|}_{\mathbf{x}}$ 174
$\mathsf{D} \tilde{\otimes} \mathsf{D}'$ 174
$\mathsf{D} \tilde{\otimes}_{\mathbf{x}}^{\mathbf{y}} \mathsf{D}'$ 174
$\mathsf{D}_1 + \mathsf{D}_2$ 49
$\mathsf{D}_1 \triangleright \mathsf{D}_2$ 49
$\mathsf{D}_1 \times \mathsf{D}_2$ 48
$\mathsf{D}_1 \uplus \mathsf{D}_2$ 65
$\mathsf{E}, \mathbf{G} \underset{c}{\overset{eq}{\longrightarrow}} \mathsf{E}', \mathbf{G}'$ 23
$\mathsf{E}|_{\mathbf{x}}$ 169
$\mathsf{E}\lightning_{k}$ 182
$\mathsf{E}_0, \mathbf{G}_0 \underset{c_1,\ldots,c_n}{\overset{eq}{\longrightarrow}} \mathsf{E}_n, \mathbf{G}_n$ 24
$\mathsf{F}|_{\mathbf{x}}$ 125
$\mathsf{T}(\Sigma)$, set of ground terms built with Σ 11
$\mathsf{T}(\Sigma, \mathsf{V})$, set of terms built with Σ and V 11
V, set of variables 11
$\mathsf{X} \underset{\mathsf{P}}{\overset{\alpha}{\longmapsto}} \mathsf{X}'$ 95
$\mathsf{X} \equiv_{\mathbb{A}} \mathsf{X}'$ 93, 175
$\mathsf{X} \underset{\mathsf{P}}{\overset{\alpha}{\longmapsto}}^{*} \mathsf{X}'$ 95
$\mathsf{X} \tilde{\triangleright} \mathsf{X}'$ 95
$\mathsf{X} \tilde{\times} \mathsf{X}'$ 95
$\mathsf{X}_1 \tilde{\uplus} \mathsf{X}_2$ 101
$\mathbb{A}$ 92, 174
$\mathcal{A}[\![\mathsf{A}]\!]$ 42
$\mathcal{A}_{\alpha}[\![\mathsf{A}]\!]$ 95
$\mathcal{B}[\![\mathbf{G}\ in\ \mathsf{P}]\!]$ 56
$\mathcal{B}[\![\mathcal{E}, \mathbf{B}\ in\ \mathsf{P}]\!]$ 170
$\mathcal{B}_{\alpha}[\![\mathsf{D}, \mathbf{B}\ in\ \mathsf{P}]\!]$ 176
$\mathcal{B}_{\alpha}[\![\mathbf{G}\ in\ \mathsf{P}]\!]$ 95
$\mathbb{C}$ 39, 168
$\tilde{\mathcal{C}}[\![\mathsf{c}]\!]$ 107
$\mathcal{C}[\![\mathsf{c}]\!]$ 42
$\mathcal{C}_{\alpha}[\![\mathsf{c}]\!]$ 95
$\mathbb{E}$ 168
$\mathbb{E}\lightning_{k}$ 182
$eqn(\vartheta)$ 14
$\mathcal{F}^{\mathcal{I}_{\alpha}}[\![\mathsf{P}_i]\!]$ 240
$\mathcal{F}_{\alpha}^{\mathcal{I}_{\alpha}}[\![\mathsf{P}_i]\!]$ 240
$\mathcal{F}[\![\mathsf{P}]\!]$ 59, 169
$\mathcal{F}_{\alpha}[\![\mathsf{P}]\!]$ 95, 175
$\Gamma(\mathsf{E})$ 180
$\Gamma(\mathsf{d})$ 142
$\Gamma(\vartheta)$ 124
$\Gamma(\mathsf{t})$ 124, 180
$\mathcal{G}[\![\mathbf{G}]\!]$ 42
$\mathcal{G}_{\alpha}[\![\mathbf{G}]\!]$ 95
$\mathbb{I}_{\mathbb{A}}$ 93, 175
$\mathbb{I}_{\mathbb{C}}$ 41, 168
$Id_{\mathbb{C}}$ 48
$Id_{\mathbb{I}}$ 48
$\mathcal{O}[\![\mathsf{P}]\!]$ 59, 170
$\mathcal{O}_{\alpha}[\![\mathsf{P}]\!]$ 95, 176
$\mathcal{OS}_{\alpha}[\![\mathsf{P}]\!]$ 205
$\mathbb{PA}$ 92
$\mathbb{PC}$ 39

Π, set of predicate symbols 11
$\mathcal{P}^{\mathcal{I}_\alpha}[\![\mathsf{P}_i]\!]$ 240
$\mathcal{P}^{\mathcal{I}_\alpha}_\alpha[\![\mathsf{P}_i]\!]$ 240
$\mathcal{P}[\![\mathsf{P}]\!]$ 42, 169
$\mathcal{P}_\alpha[\![\mathsf{P}]\!]$ 95, 175
$pu_\alpha(\mathsf{X})$ 95
$pu(\mathsf{D})$ 68
$pu^n(\mathsf{D})$ 68
$pu_n(\mathsf{D})$ 68
$\mathcal{Q}[\![\mathbf{G}\ in\ \mathsf{P}]\!]$ 42
$\mathcal{Q}_\alpha[\![\mathbf{G}\ in\ \mathsf{P}]\!]$ 95
Σ, set of function symbols 11
$su_\alpha(\mathsf{X})$ 95
$su(\mathsf{D})$ 54
$su_n(\mathsf{D})$ 61
$unf^k_{\mathsf{P},\alpha}(\mathsf{X})$ 95
$\aleph$ 3
Atoms 15
$\mathbf{G} \xrightarrow[c]{eq} \mathbf{G}'$ 170
$\mathbf{G} \xrightarrow[c]{\alpha-eq} \mathbf{G}'$ 176
$\mathbf{G} \xrightarrow[c]{\vartheta} \mathbf{B}$ 21
$\mathbf{G} \xrightarrow[P]{eq}{}^* \mathbf{G}'$ 170
$\mathbf{G} \xrightarrow[P]{\vartheta}{}^* \mathbf{B}$ 21
$\mathbf{G}_0 \xrightarrow[c_1,\ldots,c_n]{\vartheta} \mathbf{G}_n$ 21
$[\mathsf{D}]^*$ 65
$[\mathsf{X}]^*_\alpha$ 101
$\square$ 15
ε 12
Goals 15
$\phi_{\mathbf{G}}$ 48
$\lceil\Theta\rceil$ 112
$\lceil\mathcal{E}\rceil$ 178
$\mathcal{E} \equiv_x \mathcal{E}'$ 168
$\mathcal{E}|_x$ 169
$\mathcal{E}_1 \otimes \mathcal{E}_2$ 169
$\mathcal{E}_1 \otimes^x_y \mathcal{E}_2$ 169
Subst 13
$Subst\lightning_k$ 129
$use(\mathsf{P}_i)$ 233
$Ht(\mathsf{d})$ 98
$PredsDef(\mathsf{P})$ 233
$answer(\mathsf{d})$ 19
$clauses(\mathsf{d})$ 19
$elim(\mathcal{E})$ 14
$first(\mathsf{d})$ 19
$gfp\ \mathsf{f}$ 9
$last(\mathsf{d})$ 19
$length(\mathsf{d})$ 19
$lfp\ \mathsf{f}$ 9
$mgu(\mathsf{t}_1, \mathsf{t}_2)$ 13
$open(\mathsf{T})$ 97
$prefix(\mathsf{d})$ 19
$repl(\mathsf{T}, \mathsf{T}', \mathsf{T}'')$ 97
$res(\mathsf{d})$ 147
$supp(\mathsf{f})$ 3
$tree(\mathsf{c})$ 49
$wf_{\mathbf{G}}(\mathsf{S})$ 92
$\mathcal{H}_{\mathsf{D}}(\mathsf{D}')$ 65
$\mathsf{d}_1 :: \mathsf{d}_2$ 44
$\mathsf{d}_1 \wedge \mathsf{d}_2$ 45
$\mu_k(\mathsf{X})$ 129
$\partial_\delta(\mathsf{d})$ 44
ϕ 48
$\mathbf{G}\ in\ \mathsf{P}$ 15
$\mathsf{A} \mapsto \mathsf{E}$ 202
$v \mapsto r$ 4
$\sum\{\mathsf{D}_j\}_{j\in J}$ 49
$\tau_\Gamma(\mathsf{X})$ 125
$\vartheta\lightning_k$ 129
WFS 39
$\widetilde{\bigcup}\{\mathsf{D}_i\}_{i\in I}$ 174
$\widetilde{\sum}\{\mathsf{X}_i\}_{i\in I}$ 95
$\widetilde{\mathcal{H}}_{\mathsf{X}}(\mathsf{X}')$ 101
$\mathsf{f}[^v/_x]$ 4

A

abstract behavior of a query 92
abstract behavior of an abstract equational goal 176
abstract collections, $\mathbb{A}$-collections . . 92
abstract derivation 176
abstract derivation step 176
abstract diagnosis 146, 197
abstract diagnosis algorithms 234
abstract elements, $\mathbb{A}$-elements 202
abstract enhanced variance equivalence 93, 175
abstract goal 175
abstract immediate consequence operator 204
abstract interpretation 27, 90
abstract oracle 205
abstract oracle simulation 205
abstract semantic operators 72
abstract semantics 27
abstract top-down denotation 176
abstraction function 27, 28
acceptable program 212
actually-incorrect clause 228
additive functions 8
adjoints 28
AND-compositionality property . . . 25, 37, 64, 171
application of a substitution 12
approximate observables 167, 178
atom 11, 15
atomic formula 11
atomic goals 15

B

behavior of a query 56
behavior of an equational goal 170
bijective function 5
binary relation 3
binding 12
body of a clause 15
bottom element 7

C

call pattern observable η 112
chain 6
closure operator 26
co-additive function 8
co-continuous function 8
collection 39, 168
compatible extension operation $\triangleright$ . . 49
complete lattice 7
complete observables 167, 177
complete partial order 7
complete programs 200, 202
composition of substitutions 12
computed answer substitution observable ξ 93
computed answer substitutions 19
computed resultant observable χ . . . 97
concrete semantics 27
concretization function 27, 28
condensing observables 199
condensing property 64, 171
congruence relation 6
consistent set of formulas 17
constant symbols 11
continuous functions 8
correct answer observable δ 112
correct answer observable ν 178
correct answers 23
correctness of a semantics 60
correctness of an abstract operator 29

D

declarative debugging 197
denotational observables 91, 107
denotational semantics 38, 89
denumerable set 2

depth(k) computed answer observable κ 129
depth(k) computed answer observable τ_k 182
derived goal 18
diagnosis of a logic program . 200, 203
direct image of a relation 5
direct set 7
domain abstraction 92
domain of a partial function 3
domain of a substitution 12

E

eliminable variables 14
empty goal 15
empty substitution 12
enhanced variance equivalence 41, 168
equational *SLD*-derivations 170
equational clauses 16
equational derivation step 23, 170
equational derivations 23
equational goals 16, 23
equational refutations 24
equivalence relations 5
equivalent equation sets 14
extension operation $\triangleright$ 49
extensivity property 28

F

fair selection rule 21
final states of transition system $\mathcal{T}$ . . 54
finite element of a domain 92, 174
finite functions 3
finite-support functions 3
finitely failed *SLD*-tree 21
fixpoint 9
fixpoint denotation 59
formulas 11
function symbols 11

G

Galois connections 27
Galois insertions 28
goal-dependency property . 89, 91, 143
goal-independency property . . . 89, 91, 143, 145
goals 15
greatest fixpoint 9
greatest lower bound 7
ground expressions 12
ground substitutions 12
groundness analysis of computed answers 124, 180
groundness dependencies of *SLD*-tree observable Ψ 142
groundness dependencies of a substitution 124
groundness dependencies of an equation set 180
groundness dependencies of computed answer observable ψ 125
groundness dependencies of computed answer observable Υ 180

H

head of a clause 15
Herbrand base 17
Herbrand interpretation 17
Herbrand model 17
Herbrand universe 17

I

idempotent substitution 13
identity collection, $Id_{\mathbb{C}}$ 48
immediate consequence operator . . . 18
incompleteness symptoms 203
inconsistent set of formulas 17
incorrect clauses 200, 203, 204
incorrectness symptoms 203

indexed set 3
initial states of transition system $\mathcal{T}$ 54
injective functions 4
input clause 18
instance of an expression 12
instantiation operation $\cdot$ 48
intended behavior of a program . . . 200
intended declarative semantics 197
interpretations 16
interpretations in abstract domain, $\mathbb{A}$-interpretations 93, 175
interpretations in concrete domain, $\mathbb{C}$-interpretations 41, 168
inverse image of a relation 5
isomorphism 8

L

language independence property 26, 60
LD-derivations 21
least fixpoint 9
least Herbrand model 18
least upper bound 7
leftmost selection rule 21
level mappings 212
lifting of a Galois insertion 92
limit ordinal 9
locally correct semantic operators . . 29
logic programs 15
logical consequence 17
lower bound 7

M

m-incorrect clauses 234
m-uncovered $\mathbb{A}$-elements 234
m-complete programs 240
m-partially correct programs 240
m-totally correct programs 240
mgu 13
minimality of a semantics 60
model 17
modular abstract diagnosis algorithms 239
monotonic functions 8
most general unifier 13
most precise abstract behavior 93

N

non-redundant equation set 178
non-redundant substitution set . . . 112
norm for a program 213

O

observable χ, computed resultants . 97
observable δ, correct answers 112
observable η, call patterns 112
observable κ, *depth*(k) computed answers 129
observable ν, correct answers 178
observable Ψ, groundness dependencies of *SLD*-trees 142
observable ψ, groundness dependencies of computed answers 125
observable τ_{k}, *depth*(k) computed answers 182
observable Υ, groundness dependencies of computed answers 180
observable ξ, computed answer substitutions 93
observable ζ, partial proof trees 98
observables 92, 174, 199
observational equivalences 93
operation $::$ 44
operation $\cdot$ 48
operation $\triangleright$ 49
operation $\sum$, $+$ 49
operation $\times$ 48
operation $\uplus$ 65
operation $\bigwedge$ 45

operation $\widetilde{\cdot}$ 95
operation $\widetilde{\rhd}$ 95
operation $\widetilde{\times}$ 95
operation $\widetilde{\uplus}$ 101
operation $\widetilde{\sum}$, $\widetilde{+}$ 95
operational semantics 22, 89
operational semantics of a query . . . 56
operator $[\cdot]^*$ 65
operator $[\cdot]^*_\alpha$ 101
operator $\lceil\cdot\rceil$ 112, 178
operator $\mathcal{H}_{\mathrm{D}}$ 65
operator ∂_δ 44
operator pu_α 95
operator pu 68
operator pu^{n} 68
operator pu_{n} 68
operator su_α 95
operator su 54
operator su_{n} 61
operator $unf^{\mathrm{k}}_{\mathrm{P},\alpha}$ 95
operator $\widetilde{\mathcal{H}}_{\mathrm{X}}$ 101
OR-composition operation $\uplus$ 65
OR-compositionality property . 37, 65, 66, 89
ordinal numbers 9

P

p-incorrect clauses 218
p-uncovered $\mathbb{A}$-elements 218
parallel unfolding 68
partial computed answer substitutions 19
partial diagnosis 200
partial functions 3
partial orders 6
partial proof tree observable ζ 98
partial specification consistent with a specification 218
partially correct programs . . . 200, 202
partially ordered sets 6
perfect observables 90, 96
post-fixpoint 9
powerset 3
pre-fixpoint 9
pre-ordering on substitutions 13
precise semantic operators 29
precision, discussion about 72
predicate symbols 11
predicate-disjoint modules 233
predicates 15
prefix tree 21
preorders 6
primitive semantic operators 38
product operation $\times$ 48
program clauses 15
program denotations 59
programs, $\mathbb{I}_{\mathbb{C}}$-acceptables 213
pure $\mathbb{A}$-collections 92, 174
pure $\mathbb{C}$-collections 39, 168
pure atoms 15
pure goals 15
pure identity collection $Id_{\mathbb{I}}$ 48

Q

quasi-complete-join-morphisms 27
queries 15
quotient sets 6

R

range of a substitution 3, 12
refinement operators 146
relation $\xrightarrow[\mathrm{c}_1,\ldots,\mathrm{c}_{\mathrm{n}}]{\mathrm{eq}}$ 24
relation $\xrightarrow[\mathrm{c}]{\mathrm{eq}}$ 23, 170
relation $\xrightarrow[\mathrm{c}]{\alpha-\mathrm{eq}}$ 176
relation $\xrightarrow[\mathrm{c}]{\vartheta}$ 21
relation $\xrightarrow[\mathrm{c}_1,\ldots,\mathrm{c}_{\mathrm{n}}]{\vartheta}$ 21

relation $\xrightarrow[\mathsf{P}]{\mathsf{eq}}{}^*$ 170
relation $\xrightarrow[\mathsf{P}]{\vartheta}{}^*$ 21
relation $\xmapsto[\mathsf{P}]{}{}^*$ 56
relation $\xmapsto[\mathsf{P}]{\alpha}{}^*$ 95
relation $\xmapsto[\mathsf{P}]{\alpha}$ 95
relation $\xmapsto[\mathsf{P}]{}$ 54
relation composition 4
relevant modules 233
relevant unifiers 13
renamed apart 19
renamings 13
resolvents 18

S

s-semantics 24
satisfiable set of formulas 17
Schröder's elimination principle . . 125, 181
search space 21
selected atom 18, 20
selection rule 20
semi-denotational observables . 91, 115
semi-perfect observables 130
sequential unfolding 54
signature of a first order language . 15
SLD-derivations 18
SLD-refutations 19
SLD-resolution 18
SLD-trees 21
solution of an equation set 14
solved form equation sets 14
strict binary relations 6
substitutions 12
subsumption 13
success set 22
successful *SLD*-trees 21
sum operation $\sum$, + ... 49
support of a partial function 3
surjective functions 4
symptoms 202

T

terms 11
top element 7
top-down denotation 59, 170
totally correct programs 202
totally ordered sets 6
transition system $\mathcal{T}$ for concrete operational semantics 54
transitive and reflexive closure of a relation 5
tree operator *tree* 49

U

uncovered $\mathbb{A}$-elements 203, 204
uncovered elements 200
unification algorithm 13
unifiers 13
unit clauses 15
unsatisfiable set of formulas 17
upper bound 7

V

valuation 16
variable assignment 16
variables 15
variance 13
variants 13
void collection, ϕ 48

W

well-formed sets 39