Computable Set Theory and Logic Programming

Agostino Dovier
Dipartimento di Informatica,
Università di Pisa,
Corso Italia, 40 – 56100 PISA
`dovier@di.unipi.it`

Contents

Acknowledgements

I would like to thank several people for making my four years as a *studente di dottorato* a rewarding and valuable experience.

Verbose and a bit baroque in writing and (apparently) theoretically-oriented in his research the former, direct and linear in writing and application-oriented the latter, Eugenio G. Omodeo and Gianfranco Rossi crossed their academic careers at the University of Udine in the Fall of 1989. At the same time two young students were researching their Masters' theses allowing them to start a research walk. This was the beginning of the {log}[1] project: an effective usable Logic Programming language dealing with *set* entities. The longing to reach such a goal and the friendship among us have been invigorated by the difficulties related to our *gypsy* style of life imposed by academic constraints. I owe much to both of them. They taught me (or, at least, desperately tried to teach me) to measure each single word I write, to restrain my haste for submitting preliminary papers, and to persevere when the judgement of referees and of *commissari d'esame* was not the one expected. For these reasons, and many others, I wish to thank them.

People can be divided into two categories: the *constraint generators* and the *constraint solvers*. My advisor Alberto Policriti is the greatest constraint solver I have ever known. Most of the problems that this thesis deals with would probably still be *open* without our long, Saturday morning discussions. I will always remain indebted to him both for his ability to always find time for me in his full schedule and for his subtle work aimed at allowing a lasting collaboration.

The research carried on in this thesis would not have been possible

[1]Read '*setlog*'

without Giorgio Levi's confidence in my abilities and his unconditional help.

One of the most immeasurable academic rewards is co-working on one project. I have had the pleasure of co-writing papers with a number of people. Thank you Davide Aliffi, Puri Arenas-Sànchez, Paola Bruscoli, and, only lexicographically last, my old companion of many battles and dear friend, Enrico Pontelli (one of the two above-mentioned young students).

I am grateful to the *Dipartimento di Informatica dell'Università di Pisa* for the facilities it has provided to me and for the possibility to learn the basic knowledge of a lot of research fields. I am also grateful to the *Dipartimento di Matematica ed Informatica dell'Università di Udine* for providing enough hardware to allow me to conclude the thesis close to my advisor (and to my wife).

A special '*thank you*' to my colleagues Dorella Bellè, Stefano Guerrini, Marino Miculan, Stefano Mizzaro, Angelo Monti, Paola Quaglia, Alessandro Roncato, Luca Roversi, and Enea Zaffanella, for the helpful discussions and the sagacious advice in several fields.

Michael G. Seaman generously spent part of his summer'95 vacation in the (hopeless) enterprise to improve the English of my thesis. All mistakes the reader can find were added by the author after the autumn equinox.

I took advantage of the many useful comments and spurs of my referees Egon Börger, Bharat Jayaraman, Giorgio Levi, Andrea Maggiolo-Schettini, Paolo Mancarella, and Mario Rodriguez-Artalejo during the presentations of preliminary versions of my thesis. In particular, I wish to thank Mario Rodriguez-Artalejo for his *very* careful reading and proof-checking which allowed me to greatly improve the quality of the presentation. I would like to thank Gopal Gupta for setting Enrico free to work with me, although engaged in the fascinating *ACE* project.

Finally, I benefited by a lot of untraceable e-mails by colleagues from all over the world.

On a more personal level, I owe a lot to a small number of people; I prefer to settle my large debt with them in a private setting. *Non posso tuttavia non citare i miei genitori, che hanno sempre accettato le* *mie scelte economicamente incomprensibili,* and Antonella, who made my life unpredictable.

Pisa, March 1996

Chapter 1

Introduction

This dissertation is about the integration of *Computable Set Theory* and *Logic Programming*. Elementary theories of aggregates concerning lists, multi sets (in which ordering is immaterial), compact lists (in which the ordering of elements is important, while contiguous occurrences of the same element do not count), and sets are analyzed in various frameworks. Logic Programming languages dealing with such high-level data structures can be obtained as instances of the *Constraint Logic Programming* scheme and are studied in detail in the case of sets.

1.1 Why this dissertation?

Apart from meaningless answers, such as '*Why not?*' or '*Because I have to achieve my PhD degree!*', there are very good reasons which justify the researches presented in this thesis. Before explaining my personal point of view, I like to give voice to more authoritative researchers.

> *Symbolic logic was originally conceived by Leibniz, possibly in analogy with symbolic algebra, as a tool to be used computationally. This same pragmatic aim has been associated with several of the most significant periods of advance of logic during the more than two centuries that have passed since Leibniz's original suggestions: particularly with Boole's systematization of elementary propositional calculus and with the enthusiastic 1920s efforts of Hilbert and his*

school to demonstrate the full mechanizability of mathematics by solving the predicate decision problem. However, the decisive refutation of Hilbert's attempt by the famous results of Gödel and Church concerning unsolvability and undecidability pushed most later work in logic, up to our own day, in a much less pragmatic direction; in consequence of their brilliant insights, logic took on a largely negative focus, and became principally a tool for demonstrating impossibility results of all kinds.

The rise of the computer in the mid and late 1940s began to revive Leibniz's original emphasis. Propositional calculus, which had been used very successfully for some time as a computational tool for switching circuit design, proved easy to mechanize. It was also clear tat the full predicate calculus could also be mechanized (although here a semi-decision procedure was the best that one could hope for). Early expectations that techniques such as resolution would prune predicate proof searches very effectively then focused the energy and enthusiasm of artificial intelligence researchers on the possibility that a wide variety of intelligent actions could be realized by recasting them as predicate-calculus proof searches. Even though this hope has been frustrated by the exponentially large searches that plague the general predicate case even after application of the best available proof-search pruning techniques, the efforts it inspired have left behind a continuing, and steadily growing, stream of more cautiously conceived research ...

J. T. Schwartz, Foreword to [24]

It is well-known that any mathematical theory can be embedded in set theory. It hence follows that an automated theorem-prover for set theory can be used as an all-purpose *theorem prover, or as a useful component of a program that checks the correctness of mathematical proofs. Also, various specialized dialects of set theory—often dealing with finite sets only—have been designed and successfully exploited for high-level specifications of algorithms, and, more generally,* *for expressing in an unambiguous form problems that one intends to eventually submit to computers. It is therefore legitimate to expect that the automation of deduction methods for set theory will be very helpful . . . Decidability results regarding various portions of set theory have been obtained in the last ten years. The research that has led to such results aims at laying the foundation stones upon which a formal set theory, both neat and orientated towards the needs of automated deduction, can develop. . . .*

D. Cantone and E. G. Omodeo, Chapter 1 of [24]

Constraint Logic Programming (CLP) began as a natural merger of two declarative paradigms: constraint solving and logic programming. . . . the power of CLP cannot be obtained by making simple changes to LP systems. . . . The crucial insight the CLP scheme . . . was that a logic-based programming language, its operational semantics, its declarative semantics, and the relationship between these semantics could all be parametrized by a choice of domain of computation and constraints. The resulting scheme defines the class of languages $CLP(\mathcal{X})$ obtained by instantiating the parameter $\mathcal{X}$. . . . There are several . . . constraint domains of interest . . . They include . . . domain of finite sets [42] . . .

J. Jaffar and M. J. Maher, [54]

If mathematics, and hence algorithms, can be embedded in set theory, why not using sets in programming languages? Sets are, in fact, a widely-recognized data structure for specifying the requirements of a problem (cf., e.g., the specification language Z—[102]). Sets can be conveniently employed in rapid software prototyping, where the availability of high level data and operation abstraction are the key features of the implementation language. In spite of such acclaimed usefulness, only relatively few programming languages provide sets as primitive objects; among them,

- the procedural languages SETL ([96]) and SETL2 ([60]),

- the functional languages MIRANDA ([108]—it exactly provides ZF-expressions, i.e. lists defined by their property, in the style of Zermelo-Fraenkel set theory) and ME TOO ([82]),
- the deductive databases language $\mathcal{LDL}$ ([15]),
- the logic-equational languages SEL ([55]) and SuRE ([56]),
- the new general logic programming language GÖDEL ([51]), and
- the *Constraint Logic Programming* languages CLPS ([68]), CONJUNTO ([46]), and {log} ([37, 36]).

Although very useful for programming, most of the languages dealing with set entities lack in an adequate semantics definition for them; moreover, free nesting of sets is explicitly forbidden. The highly declarative nature of logic programming languages makes them particularly well-suited for extensions with set entities; in particular, the semantic analysis of such extensions is more feasible and the non-determinism inside such languages enables an easy implementation of the satisfiability algorithms for set formulae. The first proposals in this direction come from the field of *deductive databases* (see, e.g., [66, 15]). In particular, the $\mathcal{LDL}$ language, presented in [15], is a deductive database language, syntactically similar to PROLOG, and capable of handling intensional definitions of set formers, very useful for high-level queries. Operational semantics for building intensionally defined sets is bottom-up; this allows to dodge the direct facing of the set unification problem. Nevertheless, the matching problem for set terms is itself NP-complete (cf. § 4.1).

More recently, a number of papers have addressed the problem of adding set entities to logic programming in wider settings.

The logic-equational language SuRE ([56])[1] that can be seen as a natural continuation of the language SEL ([55]), has been recently introduced to provide a unifying framework which subsumes both equational and logic programming and is able to deal with set entities.

[1]SuRE stands for Subsets, Relations, and Equations, but also for the affirmative answer to the question: *Can programming be declarative and practical?*

In the preliminary definition of the language {log} ([37, 38]), extending (pure) logic programming with set entities, particular care has been taken in describing the set theory both axiomatically and model-theoretically. SLD-resolution has been modified *ad hoc* in order to capture the semantics of the entities introduced. Soundness and completeness of the extended resolution procedure are proved extending classical results; moreover, being an ad hoc extension, any simple change in the desired data structures requires to start again such tedious proofs.

Constraint Logic Programming (see the survey paper [54]) is a general framework which, once instantiated to a particular domain of computation, automatically produces an effective logic programming language whose answers can contain *constraints*. Constraints are, typically, conjunctions of positive and negative literals, but, more generally, a constraint can be any first-order formula of a given language. (In standard logic programming only positive literals, in the form of explicit substitution, are returned.) In this way it is possible to concentrate the efforts on developing algorithms for solving constraints written in a given language, namely, to check the satisfiability of them with respect to a given theory; moreover, opportune normal forms must be chosen. If the algorithms developed to perform this check satisfy a (small) number of requirements, then the extension of the SLD resolution automatically obtained guarantees soundness and completeness results. Declarativeness and efficiency are inherited from the data structure inserted in the general scheme and from the satisfiability algorithms presented, respectively.

It is not surprising that, once this concepts have become familiar to researchers, CLP languages with sets arose. In particular the language {log} has been revised from this point of view (see [42, 33] and also for the implementation [40]). Two practically oriented approaches to Constraint Logic Programming with Sets are the CLPS system ([68, 69]) and the language CONJUNTO ([46]).

The CLP scheme is hence a *parametric* language. The guideline of this thesis is a *parametric* approach to extensions of Logic Programming with aggregate (lists, multi sets, compact lists, and sets) entities. Having chosen the $CLP(\mathcal{X})$ as host framework, the efforts are concentrated on the analysis of the parameter $\mathcal{X}$. Here, and this is one of the most important steps ahead with respect to the other proposals (even the *CLP*-based, but application-oriented CLPS and CONJUNTO) is that the semantics of sets given using a first order axiomatization. Moreover, the axiomatizations of the various theories of aggregates presented is parametrically given. Namely, adding or removing one axiom, it is possible to switch from a theory to another (for instance, from sets to multi sets). In addition, the axiomatization is chosen to make natural the passage from the *pure* case (e.g. pure set theory) to the *hybrid* case—when aggregate objects can be freely combined with standard terms of Logic Programming. A careful choice of the axiomatization will reflect also in a parametric development of the unification and constraint solving algorithms for such theories. In this way, by taking care that the algorithms fulfill the *CLP* requirements, with simple modifications it is possible to obtain several *CLP* languages of which {log} is only an example. Although simple, the theories presented can be considered as *minimal* cores for more sophisticated theories of aggregates. We will adopt (cf. § 3.2) the following notations:

- `WF` and `NWF` stand for well-founded and non-well-founded, respectively;
- `lists`, `bags`, `clists`, and `sets` are abbreviations for the (minimal) theories of lists, multi sets, compact lists, and sets, respectively.

Due to the inherent difficulty of algorithmic handling of the data structure *set*, particular care has to be taken to simplify as much as possible the axiomatization (having always in mind the requirements of the Constraint Logic Programming framework). For these, as well as for historical reasons, among the various possibilities to represent theories of aggregates (for a detailed description, see § 6.1.1), we have preferred the simplest one: a list representation. The *cons* operator $[\cdot\,|\,\cdot]$ in set theoretic language is constituted by the `with` symbol $\{\cdot\,|\,\cdot\}$ whose behavior is regulated by the axiom

$$(W) \qquad x \in \{y\,|\,z\} \leftrightarrow (x \doteq y \vee x \in z)$$

With this representation, first introduced by Bernays in [17], a finite set $\{t_1, \ldots, t_n\}$ will be represented in the theory by n *element insertions* to the emptyset $\emptyset$; more concretely, by the term

$$\{t_1 \,|\, \{t_2 \,|\, \cdots \{t_n \,|\, \emptyset\} \cdots\}\}$$

In other words, the semantics associated with the binary set constructor symbol $\{\cdot \,|\, \cdot\}$ is the following:

$$\{t \,|\, s\} \quad = \quad \{t\} \cup s\,.$$

Although apparently too simple, this representation of sets enables one to depict all hereditarily finite sets (those sets whose elements are themselves hereditarily finite); in particular, it allows one to state the classical definition of numerals *à la* Von Neumann:

$$\begin{cases} \underline{0} & = & \emptyset \\ \underline{n+1} & = & \{\underline{n} \,|\, \underline{n}\} \end{cases}$$

Thanks to the list representation chosen, the usual extensionality axiom, stating when two sets are equal (cf. e.g. [64]):

$$(E) \qquad x \doteq y \leftrightarrow \forall z\,(z \in x \leftrightarrow z \in y)\ ,$$

which works in any abstract context (finite sets, as well as infinite sets of any infinite cardinality), can be replaced by more computationally oriented axioms. Such axioms will be shown to be equivalent to (E) for the set terms (i.e. terms denoting sets) expressible inside a Logic Programming Language. To give a taste of this possibility, and to guarantee that the theory provides the desired semantics for the set constructor symbol, two equational properties must hold:

$$\begin{array}{lllll} (E_1) & \forall xyz & \{x, y \,|\, z\} & \doteq & \{y, x \,|\, z\} \\ (E_2) & \forall xz & \{x, x \,|\, z\} & \doteq & \{x \,|\, z\} \end{array}$$

Axioms (E_1) and (E_2), state the *permutativity* and the (left) *absorption* properties of the set constructor symbol, respectively.

From an equational point of view, removing one (or both) from between axioms (E_1) and (E_2), the corresponding theory becomes a weaker theory (the number of objects that cannot be considered equal decreases). In particular, allowing only the permutativity property (E_1), a *multi-set* theory arises. Accepting only the absorption property (E_2), the result is a theory of lists in which contiguous occurrences of the same element are immaterial. We will refer to such entities as *compact lists*. Clearly, allowing none of them the standard *list* theory of Logic Programming is considered; with the word *aggregate* we will refer to any of the four data structures.

A first test for an axiomatization of a theory of aggregates devoted to Logic Programming is to develop a unification algorithm; unification is in fact the basic operation for the implementation of resolution; furthermore, it plays an important role in constraint solving. This is also a reasonable test from a logical point of view: considering an axiomatic set theory (e.g., `WF_sets`, our minimal theory of well-founded sets) in which the extensionality axiom (E) defined above holds, two set expressions s_1 and s_2, containing variables $y_1, \dots, y_n$, are unifiable if and only if

$$\texttt{WF\_sets} \vdash \exists y_1 \dots y_n \forall x \, (x \in s_1 \leftrightarrow x \in s_2) \, .$$

Solving the unification problem for such a theory means that the validity problem for $(\exists^* \forall)$ is decidable. In [84, 85, 87] decidability and completeness of the class of formulae $(\exists^* \forall)$ for a minimal theory of sets is presented. The theory consists of the axiom (N) stating the existence of the emptyset, the axiom (W) which regulates the behavior of the set constructor symbol, the extensionality axiom (E), and the regularity (foundation) axiom (R) (axiom (N) will be introduced in § 3.1, while axiom (R) is deeply analyzed in § A.2). This result extends the result of Gogol ([47]) which proves the completeness in ZFC of such class of formulae.[2] Any CLP interpreter should be able to decide the satisfiability of a negative literal. This purpose is logically equivalent to

$$\texttt{Sets} \vdash \exists y_1 \dots y_n \exists x \, (x \in s_1 \leftrightarrow x \notin s_2),$$

a simpler $(\exists^*)$ problem.

For these reasons, we study in all details the unification problem for all the analyzed theories of aggregates. The solutions to the constraint satisfaction problem will be also presented; particular care will be devoted to the (most difficult) set case.

[2]Notice that decidability and completeness of the class $(\exists^* \forall)$ implies decidability and completeness of the class $(\forall^* \exists)$, class proved to be undecidable in the general case of classical predicate calculus (cf. [71]).

By studying the unification problem for the various theories of aggregates, a certain number of interesting characteristics has been pointed out:

- all unification problems for theories of aggregates are NP-hard, save the list one (which is linear). Furthermore, we produce a goal-oriented unification algorithm with a polynomial complexity on a non-deterministic executor for each of the NP-hard theories proving them to be NP-complete;
- It is possible to impose that any solution to a finite conjunction of equations (Herbrand system) requires infinite aggregate objects, save for the set case. In the latter case the result in [90, 91] states that a more complex formula is needed;
- the number of most general solutions to a given unification problem for aggregate objects is always finite; nevertheless, for the multi set, compact list, and set case, such number grows very quickly. A combinatorial case-analysis for sets has been performed and an algorithm with a minimal behavior with respect to the analyzed significant examples has been designed.

In the practice of mathematics very often a set is denoted *intensionally*, by providing a condition $\varphi[x]$ that is necessary and sufficient for an element x to belong to it. This intensional definition of a set is syntactically achieved by a (standard) syntax of the form:

$$\{x \,:\, \varphi[x]\}\,.$$

Observe that, if its semantics were the desired one, the `setof` extra-logical predicate of PROLOG would be exactly equivalent to such set former (cf., e.g., [18]).

However, the introduction of intensionally defined sets in set theory requires one to accept the comprehension schema, which quickly leads to famous paradoxes such as the Russell's paradox. Although Russell paradox can easily be avoided, other pathological situations arise from the insertion of a (restricted) comprehension schema (namely the separation axiom—c.f., e.g., [64]) in the axiomatic theory. A different way to face the problems concerning the (desirable) insertion of intensional set formers in a CLP with sets framework, is the one to rewrite clauses and goals in which they appear using negation.

I would like to conclude this introductive section with a remark. While historical works on axiomatization of set theory were motivated by the purpose of incorporating mathematics inside it, the aim of the (parametric) axiomatizations presented here is only to provide simple (parametric) theories for CLP computations. For this reason, the axiomatizations presented are perhaps less interesting and elegant than the classical ones. However, I hope that the hard work done to provide a parametric and effective presentation of them will turn out to be a useful tool for programmers.

1.2 Overview of the thesis

The dissertation is divided into two parts.

The first part (Computable Set Theory for Logic Programming) is devoted to the axiomatic and model-theoretic analysis of theories of aggregates. Unification algorithms (together with their complexity analysis) for them are presented, either for the well founded case, or more in general, for the non well-founded case.

Entering into more details, Chapter 2 introduces a uniform notation for well-known concepts of first-order logic. In the first section of Chapter 3, *pure* theories of aggregates are presented. Adding or removing a few axioms, theories of lists, multi sets, compact lists, and sets are described. In the second section such axiomatizations are combined with the axiomatization for standard (free) terms of Logic Programming. Terms and universes which model them will be called *hybrid*. An interesting comparison between the anti-foundation axiom **AFA** introduced by Aczel in [3] and an axiom introduced by Maher in the context of infinite terms in [74] is performed. Moreover, formulae forcing infinite solutions to a system of equations in the aggregate theories, save the set one, are discussed.

Chapter 4 faces the unification problem for hybrid theories of aggregates defined. The choices performed to describe parametrically the axiomatizations, allow a parametric development of the unification algorithms. In § 4.1 the unification problem for the multi set, compact list, and set case are shown to be NP-hard; in § 4.2, unification algorithms *à la* Robinson are described for all the theories analyzed. They are suitable for a Logic Programming implementation; the unification algorithm for hybrid sets (presented in [37, 36]) is used by the GÖDEL language ([51]). Moreover, its implementation on Warren abstract machine presented in [40], has been useful for the implementation of the SuRE language ([56]). In § 4.3, unification algorithms for the non well founded case (that can, however, be used also for the well-founded one) are described for all the theories analyzed. In particular, showing the complexity of them, the NP-completeness of the multi-set, compact-list, and set unification problem is proved. Any set unification problem needs (in general) a large number of most general unifiers to describe the set of all possible solutions. The capability of returning (if possible) exactly such set of mgu's is an important property (the minimality property) for a unification algorithm. In other words, neither repetitions of solutions nor instances of other solutions are welcome. § 4.4 faces such problem, providing a reasonable set of sample problems for testing set unification algorithms, and briefly presenting a set unification algorithm *minimal* for all of them.

The second part (Logic Programming with Sets), is oriented to the definition of a CLP language dealing with theories of aggregates. The analysis will be performed in details for the (more difficult) set case; however, the basis for repeating such design with other theories are presented: the parametric presentation of axiomatizations makes easy this work.

In Chapter 5 algorithms for constraint handling (conjunctions of positive and negative literals on the predicate symbols $\in$ and $\doteq$) are presented. In particular the (hybrid) set case is analyzed in all details. Chapter 6 presents the CLP language {log}, an effective language dealing with hybrid sets, obtained as instance of the CLP scheme on the theory `WF_sets` described in § 3.2.4. Also intensional set definitions are allowed in {log}: the axiomatic theory is not extended with the (too strong) separation axiom (cf. end of § 1.1). We describe a translation of such entities using negation. The results obtained using Negation as Failure allow to compare the power of intensional sets of {log} with the *set grouping* of $\mathcal{LDL}$ ([15]), and with the subset-equational definitions of the language SEL ([55]); see details of such a comparison in [41]. We show how to extend Constructive Negation with set entities obtaining a wider class of accepted intensional set definitions. Chapter 7 presents some examples of the declarativeness of the language {log}. Some immediate applications for the non-well-founded set unification algorithm (equivalence and completion of automata, type finding) end the chapter.

The Appendix deepens some marginal theoretical results regarding very simple set theories, the foundation axiom, and the finiteness axiom. Then it is shown how *restricted universal quantifiers* are a purely syntactical extension of the language {log}. The last part of the Appendix refers to unification; in particular, the rewriting of an hybrid set unification problem into a pure set unification one and tables reporting the number of independent unifiers that any set unification algorithm must return to given set unification problems, are presented.

Part I

Computable Set Theory for Logic Programming

Chapter 2

Preliminaries

The aim of this chapter is mainly to give a uniform presentation of well-known concepts that are not always uniformly presented in the literature. Relevant references are cited.

2.1 First order logic

Definition 2.1 *A* FIRST ORDER LANGUAGE $\mathcal{L}$ *is determined by the set of its predicate symbols* Π, *together with the set of its functional symbols* Σ. $ar : \Pi \cup \Sigma \longrightarrow \omega$ *is said to be the* ARITY FUNCTION *and is such that* $ar(p) > 0$ *for each* p *in* Π. *A set of symbols together with its arity function* ar *is called a* SIGNATURE. *Moreover, we assume there is a denumerable set* $\mathcal{V}$*—disjoint from* Π *and* Σ*—of logical variables. The arity of variables is* 0.

Following [73] (Ch. 6), we give the following definition:

Definition 2.2 *An* ORDERED TREE *is a set* T *of lists of non-negative integers such that*
if $[a_1, \ldots, a_n, j] \in T$ *then*
$[a_1, \ldots, a_n] \in T$ *and* $[a_1, \ldots, a_n, i] \in T$, *for all* $i < j$.[1]
A TERM *over a signature* S *is a mapping* $t : T \longrightarrow S$, *where*

- $T(= dom(t))$ *is a nonempty ordered tree, and*

[1]We identify $[\,[\,a_1, \ldots, a_n\,], a_{n+1}\,]$ with $[\,a_1, \ldots, a_n, a_{n+1}\,]$, to ease notation.

- *for all ν in T, $ar(t(\nu)) = |\, \{ i \,:\, [\nu, i] \text{ in } T \} \,|$, where $[\nu, i]$ stands for the i-th son of ν.*

If S is Σ, then the term is said to be GROUND. *If S is $\Sigma \cup \mathcal{V}$, and at least one node is labeled with an element of $\mathcal{V}$, then the term is said to be non-ground.*

Definition 2.3 *Given two terms t_1, t_2 over a signature Σ, we say that t_1 is a* SUBTERM *of t_2 ($t_1 \preceq t_2$) if there exists a tuple $[a_1, \ldots, a_n]$ of non-negative integers ($n \geq 0$) such that*

$$t_1(x) \quad = \quad t_2(\mathsf{append}([a_1, \ldots, a_n], x))$$

for every $x \in dom(t_1)$. t_1 is a PROPER SUBTERM *of t_2 if $t_1 \preceq t_2$ and $t_1 \not\equiv t_2$ (i.e. exists $x \in dom(t_1)$ such that $t_1(x) \neq t_2(x)$).*

Definition 2.4 *The* HERBRAND UNIVERSE *H_Σ is defined to be the set of (ground) terms t over Σ such that $dom(t)$ is finite. The* COMPLETE HERBRAND UNIVERSE *$\bar{H}_\Sigma$ is defined to be the set of <u>all</u> terms over Σ.*

$\tau(\Sigma)$ denotes the set of (ground) terms t over Σ such that $dom(t)$ is finite, and $\tau(\Sigma \cup \mathcal{V})$ denotes the set of terms t over $\Sigma \cup \mathcal{V}$ such that $dom(t)$ is finite, respectively. Clearly, $\tau(\Sigma)$ is the same as H_Σ.

Given a term t, one can 'fold' it by fusing two nodes ν, μ of t into a single node whenever the subterms rooted at ν, μ are equivalent to each other. This will lead to a rooted multi-graph $\mathcal{G}_t$ retaining information of all essential features of t: the *picture* of t, as we name it. If there are no infinite paths in $\mathcal{G}_t$, this indicates that the original t was already finite: this is the case of an *ordinary* term. When, less demandingly, $\mathcal{G}_t$ is finite, t (which might be infinite) is said to be a *rational* term. In other words,

Definition 2.5 *A* RATIONAL TERM *is a term with finitely many different subterms.*

Generally speaking, the complete Herbrand universe is <u>not</u> constituted by algorithmic data structures. However, if one restricts one's own attention to rational terms and represents them suitably (e.g., by their graph pictures), then, assuming the signature Σ is finite, even infinite terms can be algorithmically construed and manipulated.

Definition 2.6 *A* Formula *for $\mathcal{L}(\Pi, \Sigma)$ is inductively defined as follows:*

- *$p(t_1, \dots, t_n)$, where $p \in \Pi$, $ar(p) = n$, and $t_1, \dots, t_n \in \tau(\Sigma \cup \mathcal{V})$ is a formula (atomic formula, or* Atom*, or* Positive Literal*);*
- *if φ is a formula, then $(\neg\varphi)$ is a formula (if φ is an atom, then $(\neg\varphi)$ is called a* Negative Literal*);*
- *if φ_1 and φ_2 are formulae, then $(\varphi_1 \vee \varphi_2)$ is a formula;*
- *if φ is a formula and $x \in \mathcal{V}$, then $(\exists x\, \varphi)$ is a formula.*

As usual, we will remove parentheses when they are not needed for the parsing, and, moreover, we will use the formulae $\varphi_1 \wedge \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\varphi_1 \leftarrow \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$, $\forall x\, \varphi_1$, $(\forall x \in y)(\varphi)$, and $(\exists x \in y)(\varphi)$, as a denotation for $\neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\neg\varphi_1 \vee \varphi_2$, $\varphi_2 \rightarrow \varphi_1$, $(\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$, $\neg(\exists x\, \neg\varphi_1)$, $\forall x\, (x \in y \rightarrow \varphi)$, and $\exists x\, (x \in y \wedge \varphi)$, respectively.[2]

Definition 2.7 *A* Model *for $\mathcal{L}(\Pi, \Sigma)$ is a pair $M = \langle D, I \rangle$, where*

- *D is a non-empty set, called the* Domain *of M;*
- *I is a mapping, called the* Interpretation *function, that associates:*
  - *to any constant symbol $c \in \Sigma$ some member $c^I \in D$;*
  - *to every function symbol $f \in \Sigma$, $ar(f) = n > 0$, some n-ary function $f^I : D^n \longrightarrow D$;*
  - *to every relation symbol $p \in \Pi$, $ar(p) = n > 0$, some n-ary relation $p^I \subseteq D^n$.*

For instance, given $\Pi = \{\doteq\}$ and a signature Σ containing at least one constant symbol, the Herbrand Model is defined as follows:

- the domain D is the Herbrand Universe H_Σ, the set of all ground terms generated by Σ,

[2]For the last two cases (restricted quantifiers), the membership predicate $\in$, $ar(\pi) = 2$, is assumed to belong to Π.

- $t^I = t$ for any ground term t,
- $s \doteq^I t$ if and only if s^I and t^I are syntactically equal.

We recall from [27] a few basic notions about submodels.

Definition 2.8 *Let $\mathcal{A}$ and $\mathcal{B}$ be two models of a theory T on the language $\mathcal{L}(\Pi, \Sigma)$. $\mathcal{A}$ is a* Submodel *of $\mathcal{B}$ ($\mathcal{A} \subseteq \mathcal{B}$) if*

- *$A \subseteq B$, and*
- *for any $p \in \Pi$ with $ar(p) = n$, $p^{\mathcal{A}} = p^{\mathcal{B}} \cap A^n$, and*
- *for any $f \in \Sigma$ with $ar(f) = n > 0$, $f^{\mathcal{A}} = f^{\mathcal{B}} \cap A^{n+1}$, and*
- *for any constant $c \in \Sigma$, $c^{\mathcal{A}} = c^{\mathcal{B}}$.*

$\mathcal{A}$ is an Elementary Submodel *of $\mathcal{B}$ ($\mathcal{A} \preceq \mathcal{B}$) if*

- *$\mathcal{A} \subseteq \mathcal{B}$, and*
- *for any (Π, Σ)–formula φ such that the set of free variables of φ, $FV(\varphi) = \{x_1, \ldots, x_n\}$, and for any $a_1, \ldots, a_n \in A$*

$$\mathcal{A} \models \varphi[a_1, \ldots, a_n] \quad \textit{if and only if} \quad \mathcal{B} \models \varphi[a_1, \ldots, a_n]$$

A theory T is Model-Complete *if, given two models $\mathcal{A}$ and $\mathcal{B}$ of T, $\mathcal{A} \subseteq \mathcal{B}$ implies $\mathcal{A} \preceq \mathcal{B}$.*

Definition 2.9 *When we refer to* First-Order Logic with Equality *we assume the standard equality axioms (see, e.g., [79])*

$$\begin{array}{ll} (\doteq_1) & x \doteq x \\ (\doteq_2) & x \doteq y \rightarrow (\varphi \rightarrow \varphi') \end{array}$$

where φ is any first order formula—x and y are not bounded in φ—and φ' is obtained from φ by replacing zero or more occurrences of x with y, are in the theory.

2.2 Unification and substitutions

Let $\mathcal{T}$ be an *equational theory* (namely a first-order theory whose axioms are given by universally quantified equality atoms) predicating about symbols from Σ, and let $=_{\mathrm{T}}$ be the congruence relation induced between terms by $\mathcal{T}$. It is a fundamental and straightforward result in universal algebra and unification theory (cf., e.g., [101]) that any function $\sigma : \mathcal{W} \to \tau(\Sigma \cup \mathcal{V})$ defined on a subset $\mathcal{W}$ of $\mathcal{V}$ uniquely extends into a Σ-endomorphism $\overline{\sigma}$ of $\tau(\Sigma \cup \mathcal{V})$ so that every variable belonging to $\mathcal{V} \setminus \mathcal{W}$ is fixed.

Definition 2.10 *$\overline{\sigma}$ is called a* SUBSTITUTION *if it acts as identity on all but a finite number of elements of $\mathcal{V}$. By abuse of terminology and notation, we will call σ a substitution too, and will write $s\sigma$ to indicate $\overline{\sigma}(s)$. The* COMPOSITION *of two substitutions σ and δ, denoted $\sigma\delta$, is defined as usual, to the effect that $s\sigma\delta = (s\sigma)\delta$ for all s.*

Given two substitutions μ and σ, and a subset $\mathcal{W}$ of $\mathcal{V}$, μ is said to be MORE GENERAL IN $\mathcal{T}$ *than σ with respect to $\mathcal{W}$ (in symbols, $\sigma \leq \mu$ with respect to $\mathcal{W}$) if there exists a substitution δ such that $X\sigma =_{\mathrm{T}} X\mu\delta$ for all X in $\mathcal{W}$.*

Definition 2.11 *Given $t_1, t_2 \in \tau(\Sigma \cup \mathcal{V})$, a substitution γ is said to be a $\mathcal{T}$-*SOLUTION *of $t_1 \doteq t_2$ if $t_1\gamma, t_2\gamma \in \tau(\Sigma)$, and $t_1\gamma =_{\mathrm{T}} t_2\gamma$; a substitution σ is said to be a $\mathcal{T}$-*UNIFIER *of t_1 and t_2 if the universal closure of $t_1\sigma \doteq t_2\sigma$ is a theorem of T. A $\mathcal{T}$-unifier μ of t_1 and t_2 is said to be a $\mathcal{T}$-*UNIFIER OF MAXIMAL GENERALITY *(in brief, a* UMG*) of t_1 and t_2 if for every $\mathcal{T}$-unifier σ of t_1 and t_2, $\sigma \not\leq \mu$ implies $\mu \not\leq \sigma$ with respect to $\mathcal{W}$, where $\mathcal{W}$ consists of all variables appearing in t_1 or in t_2.*

The set of all $\mathcal{T}$-unifiers of two terms s and t is denoted by $\bigcup_{\mathcal{T}}(s,t)$. A set U of $\mathcal{T}$-unifiers is said to be COMPLETE *with respect to s and t if*

$$\forall\sigma \in \textstyle\bigcup_{\mathcal{T}}(s,t)\ \exists\theta \in U\ \theta \leq_{\mathcal{T}} \sigma \textit{with respect to } FV(s) \cup FV(t).$$

A set of $\mathcal{T}$-Unifiers of maximal generality of s and t, $\mu\bigcup_{\mathcal{T}}(s,t)$ is a MINIMAL COMPLETE SET OF $\mathcal{T}$-UNIFIERS *when*

$$\forall\sigma\tau \in \mu\textstyle\bigcup_{\mathcal{T}}(s,t)\Big(\sigma \leq_{\mathcal{T}} \tau \textit{ implies } \sigma =_{\mathrm{T}} \tau\Big).$$

When the context is clear, we will omit the prefix $\mathcal{T}$ before the word unifier. Similar definitions can be given starting from a system of equations $\mathcal{E}$ instead of the unification problem $s \doteq t$.

Definition 2.12 *A* Herbrand System *is a finite conjunction $\ell_1 \doteq r_1 \wedge \ldots \wedge \ell_n \doteq r_n$ of first-order equalities, where $\ell_1, r_1, \ldots, \ell_n, r_n$ are terms over the signature $\Sigma \cup \mathcal{V}$.*

For any $\mathcal{T}$-satisfiable Herbrand system $\mathcal{E}$ involving terms from $\tau(\Sigma \cup \mathcal{V})$, a unification algorithm should be able to compute *each element* of a complete set of unifiers of $\mathcal{E}$. Notice that it is not required that it computes exactly a $\mu \bigcup_{\mathcal{T}}(\mathcal{E})$ (in general, it can not exist or be not unique). However, since for the theories $\mathcal{T}$ we will deal with in this thesis, even for simple unification problems $s \doteq t$, $\mu \bigcup_{\mathcal{T}}(s,t)$ becomes very large, a valid criterion to compare two (set) unification algorithms is the analysis of the length of the *list* of solutions computed by them (the use of the word 'list' here is needed to reflect the fact that if a unification algorithm computes exactly $\mu \bigcup_{\mathcal{T}}(s,t)$, but some solution is returned more than once, it cannot be considered minimal).

If the input system $\mathcal{E}$ is $\mathcal{T}$-unsatisfiable, any unification algorithm should conclude its computation reporting a *failure* result. The theories we will analyze in this theory are all finitary, namely $\mu \bigcup_{\mathcal{T}}(s,t)$ is always finite.

Definition 2.13 *A unification algorithm for a theory $\mathcal{T}$ is said to be* Minimal for a Unification Problem *$t_1 \doteq t_2$, if it returns exactly a minimal complete set of unifiers for it. It is said to be* Minimal *for the theory $\mathcal{T}$ if it is minimal for all unification problems written in the language of the theory $\mathcal{T}$.*

For instance, the Robinson algorithm (cf. Chapter 4), is minimal for the empty equational theory (it returns a unique—most general—unifier or `fail`).

Similar definitions can be given even when $\mathcal{T}$ is not an equational theory but a more general *first-order theory*. In this case the concept of solution must be extended to systems of equations concerning any object definable in the theory (hence, not only terms having finite domain). This allows to saying, for instance, that $\sigma = [Y/Z]$ is a the most general solution to the system $X \doteq f(X, Y) \wedge X \doteq f(X, Z)$, where X denotes the root of a rational term.

2.3 Basic decidability definitions

In this brief section we recall from [107] some basic notions that will be useful in § 6.3.7.

Definition 2.14 *A* DECISION PROCEDURE *for a given formalized theory T is a method which permits us to decide in any particular case whether a given sentence formulated in the signature of T is valid in T.*

The DECISION PROBLEM *for T is the problem of determining whether a decision procedure for T exists (and possibly of exhibiting such a procedure).*

A theory T is called DECIDABLE *or* UNDECIDABLE *according as the solution of the decision problem is positive or negative.*

A theory T is called ESSENTIALLY UNDECIDABLE *if not only T itself is undecidable, but the same holds for every consistent extension of T which has the same constants as T.*

Chapter 3

Basic Aggregate Theories

This Chapter consists of two main parts. In the first part (§ 3.1) we introduce minimal axiomatizations for (pure) membership theories, in the stream of recent computable set theory works (cf., e.g., [84, 85, 86]). Our aim is to give a uniform (parametric) presentation of

- *list theory* (§ 3.1.1), in which ordering and number of occurrences of elements are both important,
- *multi-set theory* (§ 3.1.2), in which number of occurrences of elements are important, while their position is immaterial,
- *compact-list theory* (§ 3.1.3), in which the ordering of the elements is important, while the number of consecutive occurrences of an element is immaterial,
- *set theory* (§ 3.1.4), in which, as usual, the ordering of the elements and possible repetitions of some of them are not important.

In the second part of the chapter (§ 3.2) we will integrate the axiomatization provided for pure aggregates with the axiomatization used in Logic Programming for (standard) *free* terms (cf., e.g., [74]). In other words, we will give a (minimal) axiomatization for (well-founded) terms of the form

- $f(g(a), \emptyset)$,
- $f(\{g(b), \emptyset\}, g(\emptyset))$,

- $\{\emptyset, g(\emptyset), g(g(\emptyset))\}$, etc.

and for (non-well-founded) acyclic terms defined as the (unique when opportune conditions are verified) solution to equations of the form

- $X \doteq f(X, \emptyset)$,
- $X \doteq \{X, \{\emptyset\}\}$,
- $X \doteq \{X, g(X), g(g(X)) \,|\, X\}$, etc.

Aiming at parametric definitions, particular care has been paid in choosing the axioms regulating equality between aggregates. The axioms (F_1), (E^m_k), (E^c_k), and (E^s_k)—used for lists, bags, compact lists, and sets, respectively—are bi-implications with a common left hand side. Their right hand sides consist of disjunctions of 1, 2, 3, and 4 formulas chosen from among the four of (E^s_k). Such axioms are proved to be equivalent to the *standard* ones (for instance the *extensionality* axiom in the case of sets) for all the aggregates we are interested in. Moreover, they can be used both in the pure case and in the case in which free terms are kept into account (the hybrid case).

The universe and axiomatization for well-founded hybrid sets was presented in [38, 33, 42, 36].

A preliminary analysis of the well-founded and non well-founded universe for hybrid sets and hybrid multi-sets can be found in [88]. A deep analysis of the hybrid (hyper-)sets universe and axiomatization is described in [35, 34].

3.1 Pure theories

Consider the first order theory with equality, using the set of predicate symbols $\Pi = \{\doteq, \in\}$, and consisting of the two axioms

$$\begin{array}{ll} (N) & \exists z \forall x \, (x \notin z) \\ (W) & \forall y \, v \exists w \forall x \, (x \in w \leftrightarrow x \in v \vee x \doteq y). \end{array}$$

The skolemization of (N) and (W) requires the introduction of two functional symbols, one of arity 0, denoting an ‘empty’ entity, and one of arity 2, denoting the list (multi set, compact list, set) constructor symbol. Since axioms (N) and (W) will hold for all the aggregate theories that this thesis deals with, we will introduce different functors. More precisely,

- $[\,]$ and $[\cdot\,|\,\cdot]$ for lists (cf. § 3.1.1),
- $\{\!\{\ \}\!\}$ and $\{\!\{\cdot\,|\,\cdot\}\!\}$ for multi sets (cf. § 3.1.2),
- $[\![\ ]\!]$ and $[\![\cdot\,|\,\cdot]\!]$ for compact lists (cf. § 3.1.3), and
- $\emptyset$ and $\{\cdot\,|\,\cdot\}$ for sets (cf. § 3.1.4).

For what follows we will extend standard Prolog syntactic sugar (see [73]) for denoting lists (for instance $[\,a\,|\,[\,b\,|\,[\,]\,]\,]$ will be denoted simply as $[\,a, b\,]$) to all such pairs of symbols.

Using the two functional symbols introduced above, we can write (N) and (W), in the case of lists, as

$$
\begin{array}{ll}
(N) & \forall x\,(x \notin [\,]),\ \text{and} \\
(W) & \forall y\,v\,x\,(x \in [\,y\,|\,v\,] \leftrightarrow x \in v \vee x \doteq y).
\end{array}
$$

and similarly when we want to deal with other functors. We will refer to NW for this theory, both in its function-free form and in its skolemized form.

If `L`, `M`, `C`, and `S` are reasonable theories of lists, multi sets, compact lists, and sets, respectively (we will see examples of such theories in the rest of this chapter) then the following lattices of implications hold:

$$\begin{tikzcd}
& \mathtt{M} \vdash \{\!\!\{ s_1, \ldots, s_m \}\!\!\} \doteq \{\!\!\{ t_1, \ldots, t_n \}\!\!\} \arrow[dr] & \\
\mathtt{L} \vdash [s_1, \ldots, s_m] \doteq [t_1, \ldots, t_n] \arrow[ur] \arrow[dr] & & \mathtt{S} \vdash \{s_1, \ldots, s_m\} \doteq \{t_1, \ldots, t_n\} \\
& \mathtt{C} \vdash [\![s_1, \ldots, s_m]\!] \doteq [\![t_1, \ldots, t_n]\!] \arrow[ur] &
\end{tikzcd}$$

$$\begin{tikzcd}
& \mathtt{M} \vdash \{\!\!\{ s_1, \ldots, s_m \}\!\!\} \not\doteq \{\!\!\{ t_1, \ldots, t_n \}\!\!\} \arrow[dr] & \\
\mathtt{S} \vdash \{s_1, \ldots, s_m\} \not\doteq \{t_1, \ldots, t_n\} \arrow[ur] \arrow[dr] & & \mathtt{L} \vdash [s_1, \ldots, s_m] \not\doteq [t_1, \ldots, t_n] \\
& \mathtt{C} \vdash [\![s_1, \ldots, s_m]\!] \not\doteq [\![t_1, \ldots, t_n]\!] \arrow[ur] &
\end{tikzcd}$$

where $s_1, \ldots, s_m, t_1, \ldots, t_n$ are—not necessarily distinct—ground terms. Moreover,

$$\begin{array}{lllcllll}
\mathtt{L} & \vdash & t \in [s_1, \ldots, s_m] & \leftrightarrow & \mathtt{M} & \vdash & t \in \{\!\!\{ s_1, \ldots, s_m \}\!\!\} & \leftrightarrow \\
\mathtt{C} & \vdash & t \in [\![s_1, \ldots, s_m]\!] & \leftrightarrow & \mathtt{S} & \vdash & t \in \{s_1, \ldots, s_m\}\,. &
\end{array}$$

For this reason, if the data-structure list is used in a positive result, then such result will hold also for multi sets, compact lists, and sets. Conversely, if a negative result holds for sets, it will hold for multi sets, compact lists, and lists. In the preliminary results that will follow, we will use the set constructor symbol (*NW* imply neither the importance of the ordering of the elements, nor the importance of repetitions of the same element).

The first model-theory result about *NW* is that any model must necessarily be infinite. We split the preliminaries of the proof of this important fact into a pair of technical Lemmata.

Lemma 3.1 *$NW \vdash \forall yv\, (\emptyset \neq \{\, y \,|\, v \,\})$.*

Proof. Suppose $\bar{y}$ and $\bar{v}$ are such that $NW \vdash \emptyset \doteq \{\, \bar{y} \,|\, \bar{v} \,\}$. Then, by $(\doteq_2)$ $NW \vdash \forall z(z \notin \emptyset) \rightarrow \forall z(z \notin \{\, \bar{y} \,|\, \bar{v} \,\})$. By (N), $\forall z(z \notin \emptyset)$

must hold, while (W) forces that $\forall z(z \notin \{\, \bar{y} \,|\, \bar{v} \,\})$ cannot hold, since $\bar{y} \in \{\, \bar{y} \,|\, \bar{v} \,\}$. Hence $NW \vdash$ `false`, a contradiction.

3.1 □

This means, in particular, that $NW \vdash \emptyset \neq \{\emptyset\}$, $NW \vdash \emptyset \neq \{\{\emptyset\}\}$, $NW \vdash \emptyset \neq \{\{\{\emptyset\}\}\}$, and so on.

Definition 3.2 $\{\emptyset\}^n$ *is defined, by meta-mathematical induction, as follows:*

$$\begin{cases} \{\emptyset\}^0 & = \emptyset \\ \{\emptyset\}^{n+1} & = \{\{\emptyset\}^n\} \end{cases}$$

Lemma 3.3 *For any pair of different and non-negative integers i and j, $NW \vdash \{\emptyset\}^i \neq \{\emptyset\}^j$.*

Proof. Without loss of generality, assume i to be less than j. We prove the claim by meta-mathematical induction on i.
Base) If $i = 0$ the claim follows directly from Lemma 3.1.
Step) First note that, by (N) and (W), for any non-negative k

$$(*) \qquad NW \vdash x \in \{\emptyset\}^{k+1} \leftrightarrow x \doteq \{\emptyset\}^k.$$

Assume $NW \vdash \{\emptyset\}^{i+1} \doteq \{\emptyset\}^{j+1}$. Then, by $(\doteq_2)$, using as φ the formula $(\forall z \in x)(\{\emptyset\}^i \doteq z)$ we would have $NW \vdash (\forall z \in \{\emptyset\}^{i+1})(\{\emptyset\}^i \doteq z) \to (\forall z \in \{\emptyset\}^{j+1})(\{\emptyset\}^i \doteq z)$.

By $(*)$, this is equivalent to saying that $NW \vdash$ `true` $\to \{\emptyset\}^i \doteq \{\emptyset\}^j$, which contradicts the induction hypothesis, unless NW is inconsistent. Lemma 3.1 ensures that this is not the case.

3.3 □

Lemma 3.3 states in particular that any model of NW must model all the different elements $\{\emptyset\}^i$ with different objects, for any non-negative integer i. Hence

Theorem 3.4 *Any model of NW is infinite.* □

The next fact we will analyze is that the theory NW is not complete (nor model-complete). First, we analyze some interesting models of NW (cf. Def. 2.7).

- The Herbrand model $H_\Sigma = \langle \tau(\Sigma), id_\Sigma \rangle$, where for each ground term $t \in \tau(\Sigma)$, $t^{id_\Sigma} = t$; moreover, $\doteq^{id_\Sigma}$ is the syntactical equality between terms, $\in^{id_\Sigma} (t, s)$ is `true` if and only if s is of the form $\{\cdots, t, \cdots\}$.
- Let $\equiv$ be a congruence relation on $\tau(\Sigma)$. For each $t \in \tau(\Sigma)$, with $[t]_\equiv$ we denote the (canonical) representative of the equivalence class modulo $\equiv$ in $\tau(\Sigma)$.

  With $H_\Sigma / \equiv$ we denote the structure $\langle \{[t]_\equiv : t \in \tau(\Sigma)\}, I \rangle$, where $t^I = [t]_\equiv$, for any term t, $t_1 \doteq^I t_2$ is `true` if and only if $[t_1]_\equiv = [t_2]_\equiv$, and $t_1 \in^I t_2$ is `true` if and only if $[t_2]_\equiv$ is of the form $\{\cdots, [t_1]_\equiv, \cdots\}$.

  In particular, if the following (E_1) and (E_2) are the definitions of the permutativity property and of the absorption property:

  $$\begin{array}{lrcl} (E_1) & \{\, x, y \,|\, z \,\} & \doteq & \{\, y, x \,|\, z \,\} \\ (E_2) & \{\, x, x \,|\, z \,\} & \doteq & \{\, x \,|\, z \,\}, \end{array}$$

  then the structures

  – $H_\Sigma / \equiv_m$, induced by (E_1) (m here stands for *multi sets*),

  – $H_\Sigma / \equiv_c$, induced by (E_2), (c here stands for *compact lists*), and

  – $H_\Sigma / \equiv_s$, induced by (E_1) and (E_2) (s here stands for *sets*)

  are all models of NW. An aggregate oblect is said to be hereditarily finite if it is finite—namely the number of (occurrences of) its elements is finite—and every its element is hereditarily finite. The domains of the models just described represent the set of the Hereditarily Finite and Well-founded (there cannot be infinte descending chains of membership) multi-sets, compact-lists, and sets, respectively.
- We denote as $H_{\Sigma \cup \{*\}}$, where '$*$' is a constant symbol which is not in Σ, the model $\langle \tau(\Sigma) \cup \{*\}, id_\Sigma \rangle$.
- The universe of all infinite trees ($\mathcal{IT}$—or complete Herbrand Universe—cf. Def. 2.4) over the signature Σ.

Definition 3.5 *In agreement with the literature, we define as* Domain Closure Axiom *(briefly denoted as* (DCA)*): the axiom*

$$(DCA) \qquad \forall x \bigvee_{f \in \Sigma} \exists x_1 \dots x_{ar(f)}\, x = f(x_1, \dots, x_{ar(f)}).$$

Observe that if Σ is infinite, *(DCA)* is not a first-order axiom; when $\Sigma = \{\emptyset, \{\cdot \,|\, \cdot\}\}$, it becomes the formula

$\forall x\, (x \doteq \emptyset \vee \exists y v\, (x = \{\, y \,|\, v \,\}))$.

Observe that

- $H_\Sigma \models \{\,\emptyset, \{\,\emptyset\,\}\,\} \not\doteq \{\,\{\,\emptyset\,\}, \emptyset\,\}$,
  $H_\Sigma \models \{\,\emptyset, \emptyset\,\} \not\doteq \{\,\emptyset\,\}$,
  $H_\Sigma \models (DCA)$,
  $H_\Sigma \models \forall x\, (x \not\doteq \{\,x\,\})$,
  $H_\Sigma \models \forall x\, (x \not\doteq \{\,\emptyset \,|\, x\,\})$.

- $H_\Sigma/\equiv_m \models \{\,\emptyset, \{\,\emptyset\,\}\,\} \doteq \{\,\{\,\emptyset\,\}, \emptyset\,\}$,
  $H_\Sigma/\equiv_m \models \{\,\emptyset, \emptyset\,\} \not\doteq \{\,\emptyset\,\}$,
  $H_\Sigma/\equiv_m \models (DCA)$,
  $H_\Sigma/\equiv_m \models \forall x\, (x \not\doteq \{\,x\,\})$,
  $H_\Sigma/\equiv_m \models \forall x\, (x \not\doteq \{\,\emptyset \,|\, x\,\})$.

- $H_\Sigma/\equiv_c \models \{\,\emptyset, \{\,\emptyset\,\}\,\} \not\doteq \{\,\{\,\emptyset\,\}, \emptyset\,\}$,
  $H_\Sigma/\equiv_c \models \{\,\emptyset, \emptyset\,\} \doteq \{\,\emptyset\,\}$,
  $H_\Sigma/\equiv_c \models (DCA)$,
  $H_\Sigma/\equiv_c \models \forall x\, (x \not\doteq \{\,x\,\})$,
  $H_\Sigma/\equiv_c \models \exists x\, (x \doteq \{\,\emptyset \,|\, x\,\})$.

- $H_\Sigma/\equiv_s \models \{\,\emptyset, \{\,\emptyset\,\}\,\} \doteq \{\,\{\,\emptyset\,\}, \emptyset\,\}$,
  $H_\Sigma/\equiv_s \models \{\,\emptyset, \emptyset\,\} \doteq \{\,\emptyset\,\}$,
  $H_\Sigma/\equiv_s \models (DCA)$,
  $H_\Sigma/\equiv_s \models \forall x\, (x \not\doteq \{\,x\,\})$,
  $H_\Sigma/\equiv_s \models \exists x\, (x \doteq \{\,\emptyset \,|\, x\,\})$.

- $H_{\Sigma \cup \{*\}} \models \{\,\emptyset, \{\,\emptyset\,\}\,\} \not\doteq \{\,\{\,\emptyset\,\}, \emptyset\,\}$,
  $H_{\Sigma \cup \{*\}} \models \{\,\emptyset, \emptyset\,\} \not\doteq \{\,\emptyset\,\}$,
  $H_{\Sigma \cup \{*\}} \not\models (DCA)$.

- $\mathcal{IT} \models \{\emptyset, \{\emptyset\}\} \neq \{\{\emptyset\}, \emptyset\}$,
  $\mathcal{IT} \models \exists x\,(x \doteq \{x\})$,
  $\mathcal{IT} \models \exists x\,(x \doteq \{\emptyset \mid x\})$.

Given a theory T and two models $\mathcal{A}$ and $\mathcal{B}$ of T such that $\mathcal{A} \subseteq \mathcal{B}$, it is easy to prove (see Def. 2.8 and [27]) that $\mathcal{A} \preceq \mathcal{B}$ if and only if for any formula of the form $\exists x\,\varphi(x, x_1, \dots, x_n)$ and for any $a_1, \dots, a_n \in A$, if $\mathcal{B} \models \exists x\,\varphi(x, a_1, \dots, a_n)$ then there exists an $a \in A$ such that $\mathcal{A} \models \varphi(a, a_1, \dots, a_n)$. Then

Theorem 3.6 *NW is neither a complete nor a model-complete theory.*

Proof. *NW* is not complete: $H_\Sigma \models \{\emptyset, \emptyset\} \neq \{\emptyset\}$ and $H_\Sigma / \equiv_s \models \{\emptyset, \emptyset\} \doteq \{\emptyset\}$.
For the model-completeness, consider the two models H_Σ and $H_{\Sigma \cup \{*\}}$. Clearly, $H_\Sigma \subseteq H_{\Sigma \cup \{*\}}$; let φ be the formula $\forall y (y \notin x_1) \wedge x_1 \neq x_2$. It is easy to see that $H_\Sigma \not\models \exists x_1\,\varphi[x_1, \emptyset]$, while $H_{\Sigma \cup \{*\}} \models \varphi[*, \emptyset]$. Hence $H_\Sigma \not\preceq H_{\Sigma \cup \{*\}}$.

3.6 □

Undecidability Results

It has been shown in several papers (e.g. [110, 89, 16, 80]) that the theory *NW* is not decidable. For instance, in [110] Vaught showed the essential undecidability (see Def. 2.14) of *NW* (therein called Z_1). In [16] the undecidability of *NW* has been proved directly, via reduction to the halting problem. In particular it is shown that

Theorem 3.7 (Bellé, Parlamento) *It is undecidable whether or not a sentence*

$$\forall x\,(\ell_1 \doteq r_1 \wedge \cdots \wedge \ell_k \doteq r_k \wedge \ell_{k+1} \neq r_{k+1})$$

where ℓ_i's and r_i's are $(\Sigma \cup \{x\})$-terms, is satisfiable in NW.

3.1.1 Lists

In this section we will add new axioms to NW so as to model exactly a theory of lists; to this aim, we start from $\Pi = \{\doteq, \in\}$ and $\Sigma = \{[\,], [\cdot \mid \cdot]\}$. Among the models for NW presented, the models that better reflects the semantics of a list constructor (namely, both ordering and repetitions of elements are relevant) are the Herbrand model H_Σ and the complete Herbrand model $\mathcal{IT}$. Therefore, we need an axiomatization of such universes. This has been done in literature by introducing the so-called freeness axioms—see, for instance, [75, 28], in a simpler signature not comprising the membership symbol $\in$.

The first two freeness axioms, in our signature, became simply

$$
\begin{array}{ll}
(F_1) & \forall y_1 y_2 v_1 v_2 \, ([\, y_1 \mid v_1 \,] \doteq [\, y_2 \mid v_2 \,] \rightarrow y_1 \doteq y_2 \wedge v_1 \doteq v_2) \\
(F_2) & \forall y v \, ([\,] \not\doteq [\, y \mid v \,]).
\end{array}
$$

Notice that

1. $H_\Sigma \models (F_1)$ and $\mathcal{IT} \models (F_1)$, while $H_\Sigma / \equiv_m \not\models (F_1)$, $H_\Sigma / \equiv_c \not\models (F_1)$, and $H_\Sigma / \equiv_s \not\models (F_1)$.

2. $NW \vdash (F_2)$, as proved in Lemma 3.1.

Thus, the behavior of membership allows us to forget axiom (F_2), which is a theorem.

Moreover, the third freeness axiom, which is an axiom schema,

$$
(F_3) \qquad \forall x \, (x \not\doteq t[x])
$$

(where $t[x]$ is any term containing x, except x itself) is a sort of syntactic version of the regularity axiom, as will be shown in detail in § A.2, and it is used to ensure that any model of the theory contains an isomorphical copy of the Herbrand model. Notice that

1. $H_\Sigma \models (F_3)$ and $H_\Sigma / \equiv_m \not\models (F_3)$, while

2. $H_\Sigma / \equiv_s \not\models (F_3)$, since $H_\Sigma / \equiv_s \models [\,[\,]\,] \doteq [\,[\,], [\,]\,]$, and $\mathcal{IT} \not\models (F_3)$, since $x = [\, x \,]$ is solvable in $\mathcal{IT}$.

The insertion of axiom schema (F_3) in the theory allows to describe well-founded lists only; as it will be clarified in § A.2, in particular it states that, for any n, there are no elements $x_0, \ldots, x_n$ such that

$x_0 \in x_1 \wedge x_1 \in x_2 \wedge \cdots \wedge x_{n-1} \in x_n \wedge x_n \in x_0$.

Theorems 3.6 and 3.7 point out the difficulty in managing *NW* algorithmically. Moreover, we will see that the theory we present here is complete and decidable for a significant class of sentences.

In [74], the *completeness* of the theory consisting of the axioms ($\doteq_1$), ($\doteq_2$), (F_1), (F_2), (F_3), and (DCA), when $\Pi = \{\doteq\}$ and Σ is any finite signature, has been proved. For infinite signatures, (DCA) is not needed for completeness.[1] ($\in$ is not a predicate symbol of the language analyzed.) The proof of this fact is given by a *quantifier elimination procedure* that can be applied unaltered to our theory $NWF_1F_3(DCA)$, when in the formula there are no occurrences of the predicate symbol '$\in$'. Such result is implicitly a proof of the following

Theorem 3.8 *If φ is a $(\{\doteq\}, \Sigma)$-formula (i.e. no occurrences of $\in$ are in φ) then:*

$NWF_1F3(DCA) \vdash \varphi$ *is decidable.*

Moreover, such theory is complete for the above class of formulae.

Nevertheless, the essential undecidability of *NW* stated in [110], implies that the theory constituted by the axioms $NWF_1F3(DCA)$ is undecidable, namely it is impossible to extend the result of Theorem 3.8 to any $(\{\doteq, \in\}, \Sigma)$-formula. An example of the difficulty to extend such result is the following: in § 3.1.2 it will be shown that, if the permutativity property held for $[\,\cdot\,|\,\cdot\,]$, then $t \in s$ would be equivalent to $\exists v\,(s \doteq [\,t\,|\,v\,])$, for any *finite* list s. However, since such property does not hold for $NWF_1F3(DCA)$ (the co-existence of the permutativity property with axiom (F_1) is inconsistent), it would be impossible to rewrite a membership literal into a simple equality formula. In particular, for any finite list s,

$$x \in s \quad \leftrightarrow \quad \exists z_0 \cdots z_n\,(v \doteq [\,z_1, \ldots, z_n, x\,|\,z_0\,])$$

but the value of n is not known.

[1] If $\Sigma \supset \{[\,], [\cdot\,|\,\cdot]\}$, then axioms ($F_1$) and ($F_2$) must be generalized to all elements of Σ, as it will be done in § 3.2.

The non-well-founded theory of lists (in which axiom schema (F_3) is replaced by a suitable axiom) will be presented and analyzed in a stronger framework in § 3.2.1. Also in this case the result of Maher will hold (see [74] for the proof).

As a side remark, usually, a list theory supports the functional symbols *car* and *cdr*, whose semantics is the following

$$car([\,x\,|\,y\,]) \;\doteq\; x \qquad\qquad cdr([\,x\,|\,y\,]) \;\doteq\; y\,.$$

We can introduce such operators into the presented theory simply by skolemizing axiom (DCA):

$$(DCA) \qquad \forall x\,(x \not\doteq [\,] \rightarrow x \doteq [\,car(x)\,|\,cdr(x)\,])\,.$$

In [100] an elegant unification algorithm (based on a rewriting into a normal form) for lists with *car* and *cdr* is presented.

3.1.2 Multi sets

Adding to the theory NW over the language $\Pi = \{\doteq, \in\}$ and $\Sigma = \{\{\!\!\{\ \}\!\!\}, \{\!\!\{\cdot\,|\,\cdot\}\!\!\}\}$, the axiom

$$(E_1) \qquad \forall xyz\,\{\!\!\{\,x, y\,|\,z\,\}\!\!\} \doteq \{\!\!\{\,y, x\,|\,z\,\}\!\!\}$$

we define the theory NWE_1, a minimal theory of multi sets.

In literature, multi sets are usually called *bags* because of their intended meaning: two housewives consent to swap their shopping bags only if they contain exactly the same sundries. The position of the sundries in the bag is immaterial, while the number of each item in the bag is important.

We will present some basic results about bags.

Lemma 3.9 *Let $\pi : \{1, \dots, n\} \longrightarrow \{1, \dots, n\}$ be a permutation. Then*

$$E_1 \;\vdash\; \{\!\!\{\,s_1, \dots, s_n\,\}\!\!\} \doteq \{\!\!\{\,s_{\pi_1}, \dots, s_{\pi_n}\,\}\!\!\}$$

for any n-tuple of Σ-terms $s_1, \dots, s_n$.

Proof. First we show the 'swap' property:

$$E_1 \vdash \{\!\!\{\, a_1, \dots, a_h, s, b_1, \dots, b_k, t \,|\, r \,\}\!\!\} \doteq \{\!\!\{\, a_1, \dots, a_h, t, b_1, \dots, b_k, s \,|\, r \,\}\!\!\}$$

provided a_i, b_j, s, t, and r be Σ-terms.

$$\begin{array}{lllll}
E_1 & \vdash & \{\!\!\{\, s, b_1, \dots, b_k, t \,|\, r \,\}\!\!\} & \doteq & \text{by } (E_1) \\
 & & \{\!\!\{\, b_1, s, \dots, b_k, t \,|\, r \,\}\!\!\} & \doteq & \text{by } (E_1) \text{ and } (\doteq_2) \\
 & & \vdots & \vdots & \vdots \quad \vdots \\
 & & \{\!\!\{\, b_1, \dots, b_k, s, t \,|\, r \,\}\!\!\} & \doteq & \text{by } (E_1) \text{ and } (\doteq_2) \\
 & & \{\!\!\{\, b_1, \dots, b_k, t, s \,|\, r \,\}\!\!\} & \doteq & \text{by } (E_1) \text{ and } (\doteq_2) \\
 & & \vdots & \vdots & \vdots \quad \vdots \\
 & & \{\!\!\{\, b_1, t, \dots, b_k, s \,|\, r \,\}\!\!\} & \doteq & \text{by } (E_1) \\
 & & \{\!\!\{\, t, b_1, \dots, b_k, s \,|\, r \,\}\!\!\} & &
\end{array}$$

The swap property follows immediately for $(\doteq_2)$.

Since the permutation π is known, it is easy to write an algorithm which, performing at most n 'swaps', rewrites $\{\!\!\{\, s_1, \dots, s_n \,\}\!\!\}$ as $\{\!\!\{\, s_{\pi_1}, \dots, s_{\pi_n} \,\}\!\!\}$.

3.9 □

Theorem 3.10 $E_1 \vdash \{\!\!\{\, s_1, \dots s_m \,\}\!\!\} \doteq \{\!\!\{\, t_1, \dots t_n \,\}\!\!\}$ *if and only if* $m = n$ *and* $E_1 \vdash t_i \doteq s_{\pi_i}$ *for some permutation* $\pi : \{1, \dots, n\} \longrightarrow \{1, \dots, n\}$.

Proof. The $(\leftarrow)$ direction follows from equality and Lemma 3.9. Under E_1, $a \equiv \{\!\!\{\, s_1, \dots s_m \,\}\!\!\}$ can be proved to be equal to $b \equiv \{\!\!\{\, t_1, \dots t_n \,\}\!\!\}$ if (and only if) a can be re-written to b using the following two rules:

$$\begin{array}{rrcl}
(\doteq_2)^{inst} & x \doteq x' \wedge z \doteq z' & \Rightarrow & \{\!\!\{\, x \,|\, z \,\}\!\!\} \doteq \{\!\!\{\, x' \,|\, z' \,\}\!\!\} \\
(E_1) & x \doteq x' \wedge y \doteq y' \wedge z \doteq z' & \Rightarrow & \{\!\!\{\, x, y \,|\, z \,\}\!\!\} \doteq \{\!\!\{\, y', x' \,|\, z' \,\}\!\!\}
\end{array}$$

It is immediate to see that a necessary condition is that $m = n$. Moreover, if s_i is different from all $t_1, \dots t_n$ (or *viceversa*), then such rewriting is impossible.

3.10 □

Models for multi sets

Clearly, any model of NWE_1 is also a model of NW; an intuitive model for NWE_1 is the model $H_\Sigma/\equiv_m$, which was briefly touched on in § 3.1. We will first analyze such model and later modify the axiomatization in order to capture as much as possible the characteristics of such model.

We describe now a way for choosing the canonical representative $[\,t\,]$ of any term t in H_Σ. Consider the well-ordering $<_\Sigma$ on H_Σ described as follows:

- $\{\!\{\ \}\!\} <_\Sigma \{\!\{\, y \,|\, v \,\}\!\}$, for any y and v in H_Σ;
- $\{\!\{\, y_1 \,|\, v_1 \,\}\!\} <_\Sigma \{\!\{\, y_2 \,|\, v_2 \,\}\!\}$ if $(y_1 <_\Sigma y_2)$ or $(y_1 \equiv y_2$ and $v_1 <_\Sigma v_2)$.

Consider a finite multiset $\{\!\{\, s_1, \dots, s_n \,\}\!\}$. From Theorem 3.10 it follows that $\leq n! \times \mid [\, s_1 \,] \mid \times \cdots \times \mid [\, s_n \,] \mid$
(= if and only if the $[\, s_i \,]$'s are pairwise distinct) elements belong to $[\, \{\!\{\, s_1, \dots, s_n \,\}\!\} \,]$. The canonical representative will be the one preceding all the others with respect to $<_\Sigma$.

For instance, the canonical representative of

$$\{\!\{\{\!\{\{\!\{\ \}\!\},\{\!\{\{\!\{\ \}\!\}\}\!\}\}\!\},\{\!\{\ \}\!\},\{\!\{\{\!\{\{\!\{\{\!\{\ \}\!\}\}\!\}\}\!\},\{\!\{\ \}\!\},\{\!\{\ \}\!\}\}\!\}\}\!\}$$

is

$$\{\!\{\{\!\{\ \}\!\},\{\!\{\{\!\{\ \}\!\},\{\!\{\ \}\!\},\{\!\{\{\!\{\{\!\{\ \}\!\}\}\!\}\}\!\}\}\!\},\{\!\{\{\!\{\ \}\!\},\{\!\{\{\!\{\ \}\!\}\}\!\}\}\!\}\}\!\}.$$

We define here one enumeration function α_m of $H_\Sigma/\equiv_m$. Such function is defined on any Σ-term; however, as it will be proved, it takes the same value for all members of an $\equiv_m$ equivalence class. Let $p_1, p_2, p_3, \dots$ be the first, second, third, ... prime number.

$$\begin{cases} \alpha_m(\{\!\{\ \}\!\}) & = \quad 1 \\ \alpha_m(\{\!\{\, s_1, \dots, s_n \,\}\!\}) & = \quad \prod_{i=1}^{n} p_{\alpha_m(s_i)} \end{cases}$$

Lemma 3.11 *$\alpha_m : H_\Sigma \to \omega \setminus \{0\}$ is onto. Furthermore, $\alpha_m(t_1) = \alpha_m(t_2)$ implies $E_1 \vdash t_1 \doteq t_2$.*

Proof. By induction on k we show first that for all $k > 0$ exists $t \in H_\Sigma$ such that $\alpha_m(t) = k$.
Base) $\alpha_m(\{\!\{\ \}\!\}) = 1$ by definition.

Step) Suppose the claim is true for any $k \leq h$. By the (unique) factorization Theorem $\exists! \beta_1 \ldots \beta_r \, (h+1 = p_{i_1}^{\beta_1} \times \cdots \times p_{i_r}^{\beta_r})$ with $\beta_i > 0$. Since $x < p_x$ for any $x \in \omega \setminus \{0\}$, by induction hypothesis exist $s_1, \ldots, s_r$ such that $\alpha_m(s_1) = i_1$ and ... and $\alpha_m(s_r) = i_r$.
By definition, $\alpha_m(\{\!\!\{ s_1, \ldots, s_r \}\!\!\}) = h+1$.

We prove α_m is a semantical injection by induction on *rank*(x) (where the function *rank* is defined *rank*$(\{\!\!\{\ \}\!\!\}) = 0$, *rank*$(\{\!\!\{ y \mid v \}\!\!\}) = \max\{1 + \textit{rank}(y), \textit{rank}(v)\})$.
Base) $\{\!\!\{\ \}\!\!\}$ is the only element whose *rank* is 0.
Step) Suppose the claim is true for multi-sets of *rank* less than or equal to h. Consider $\{\!\!\{ s_1, \ldots, s_m \}\!\!\}$ and $\{\!\!\{ t_1, \ldots, t_n \}\!\!\}$ such that

$$\textit{rank}(\{\!\!\{ s_1, \ldots, s_m \}\!\!\}) = h+1 \text{ and } \textit{rank}(\{\!\!\{ t_1, \ldots, t_n \}\!\!\}) \leq h+1 \, .$$

This means in particular that *rank*(s_i), *rank*$(t_j) \leq h$, for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.
Assume $\alpha_m(\{\!\!\{ s_1, \ldots, s_m \}\!\!\}) = \alpha_m(\{\!\!\{ t_1, \ldots, t_n \}\!\!\})$. By the unique factorization Theorem, for any integer k

$$\mid S_k = \{s_i : \alpha_m(s_i) = k\} \mid = \mid T_k = \{t_j : \alpha_m(s_j) = k\} \mid \, .$$

Consider a value of k such that S_k is not empty. By induction hypothesis, all the elements of $S_k \cup T_k$ are equivalent for E_1. This means that we may replace all occurrences of them in $\{\!\!\{ s_1, \ldots, s_m \}\!\!\}$ and $\{\!\!\{ t_1, \ldots, t_n \}\!\!\}$ with a unique representative (not necessarily the canonical one). By Lemma 3.9, $E_1 \vdash \{\!\!\{ [s_1], \ldots, [s_m] \}\!\!\} \doteq \{\!\!\{ [t_1], \ldots, [t_n] \}\!\!\}$.

3.11 □

Corollary 3.12 *Given two Σ-terms t_1 and t_2, $E_1 \vdash t_1 \doteq t_2$ if and only if $\alpha_m(t_1) = \alpha_m(t_2)$.* □

This suggests a quite simple numerical procedure for solving the unification problem of two ground terms in the theory NWE_1.

In figure 3.1 we present a Prolog program which returns the canonical representative of a ground term (predicate canon) and the value of α_m for it (predicate get_value).
(The auxiliary predicate prime(N, P) returns in P the N^{th} prime number.)

```
canon({[ ]}, {[ ]}).
canon({[ T | S ]}, Can) :-
    canon(T, Tcan),
    canon(S, Scan),
    ev_insert(Tcan, Scan, Can).
ev_insert(T, {[ ]}, {[ T ]}).
ev_insert(T, {[ S | R ]}, {[ T, S | R ]}) :-
    get_value(T, ValT),
    get_value(S, ValS),
    ValT ≤ ValS,
    !.
ev_insert(T, {[ S | R ]}, {[ S | R1 ]}) :-
    ev_insert(T, R, R1).

get_value({[ ]}, 1).
get_value({[ T | S ]}, Val) :-
    get_value(T, ValT),
    get_value(S, ValS),
    prime(ValT, P),
    Val is (P * ValS).
```

Figure 3.1: Reducing bags to canonical form

Equality criterion for bags

We are going to find a bridge between equality and membership; for doing that we will introduce different equality criterion for bags. Their relations under some conditions will be proved.

It is consistent for the theory NWE_1 that $\{\!\{\, \{\!\{\ \}\!\}, \{\!\{\ \}\!\} \,\}\!\} \doteq \{\!\{\, \{\!\{\ \}\!\} \,\}\!\}$ (it is true in the model $H_\Sigma / \equiv_s$). We need to enforce the theory in order to make the data-structure multi-set distinct from the set one. First we analyze the following

Lemma 3.13 *For all integer n and for all $x, y_1, \ldots, y_n$*

$$NWE_1 \;\vdash\; x \in \{\!\{\, y_1, \ldots, y_n \,\}\!\} \leftrightarrow \exists z\, (\{\!\{\, y_1, \ldots, y_n \,\}\!\} \doteq \{\!\{\, x \,|\, z \,\}\!\}).$$

Proof. The *if* direction follows trivially from (W).
For the *only if* direction, by n applications of (W) and one of (N).

$$NWE_1 \vdash x \in \{\!\{\, y_1, \ldots, y_n \,\}\!\} \quad \text{if and only if} \quad x \doteq y_1 \vee \ldots \vee x \doteq y_n.$$

Let $x \doteq y_i$ be the (not necessarily unique) true disjunct. By Lemma 3.9,

$$NWE_1 \;\vdash\; \{\!\{\, y_i, y_1, \ldots, y_{i-1}, y_{i+1}, \ldots y_n \,\}\!\} \doteq \{\!\{\, y_1, \ldots, y_n \,\}\!\}.$$

Choose z as $\{\!\{ y_1, \ldots, y_{i-1}, y_{i+1}, \ldots y_n \}\!\}$.

3.13 □

As corollary, after giving the following intuitive definition of finiteness:

- a multiset m is said to be *finite* if it is $\{\!\{\ \}\!\}$ or if there exists an integer n such that m is written as $\{\!\{ y_1, \ldots, y_n \}\!\}$ for some $y_1, \ldots, y_n$,[2]

we have:

Corollary 3.14 *If v is a finite multiset, then*

$$NWE_1 \ \vdash \ \forall x \Big(x \in v \leftrightarrow \exists z \, (v \doteq \{\!\{ x \mid z \}\!\}) \Big).$$

The finiteness requirement is a necessary conditions for the above corollary. This will follow from the following remark.

Remark 3.15 *We will define a model $M = \langle D, I \rangle$ of NWE_1 in which there is an object v containing infinite elements and such that the condition of Corollary 3.14 does not hold for v.*

Consider the domain D_H of the model $H_\Sigma / \equiv_m$ defined in § 3.1 (namely, the set of all hereditarily finite and well-founded bags). The domain D is obtained extending D_H with the object v fulfilling the follow properties:

$v = \{\!\{ \{\!\{\ \}\!\} \mid v \}\!\}$, *and* $\{\!\{ \{\!\{\ \}\!\} \}\!\} \in v$ *exactly once.*

Assume now that $x \prec v$ for all $x \in D_H$, in order to model axiom (W), we must insert in D a representative for all the objects obtained by applying a finite number of times the $\{\!\{ \cdot \mid \cdot \}\!\}$ operator to the elements of D_H and v. ($\prec$ relation is used to choose the representative). The element v is such that $\{\!\{ \{\!\{\ \}\!\} \}\!\} \in v$, but there is no z in D such that $v \doteq \{\!\{ \{\!\{ \{\!\{\ \}\!\} \}\!\} \mid z \}\!\}$.

The aim of the following propositions is to analyze the existing relations among different equality criteria for multisets. We start from a trivial lemma whose proof is omitted.

[2]For a more formal definition of finiteness see § A.3.

Lemma 3.16

$$\begin{array}{rrl} (i) & (\doteq) \vdash & \forall y_1 y_2 v_1 v_2 (\ \ y_1 \doteq y_2 \land v_1 \doteq v_2 \to \\ & & \qquad \{\!\!\{ y_1 \mid v_1 \}\!\!\} \doteq \{\!\!\{ y_2 \mid v_2 \}\!\!\} \ ), \\ (ii) & (\doteq)(E_1) \vdash & \forall y_1 y_2 v_1 v_2 (\ \ \exists z \, (v_1 \doteq \{\!\!\{ y_2 \mid z \}\!\!\} \land v_2 \doteq \{\!\!\{ y_1 \mid z \}\!\!\}) \to \\ & & \qquad \{\!\!\{ y_1 \mid v_1 \}\!\!\} \doteq \{\!\!\{ y_2 \mid v_2 \}\!\!\} \ ), \\ (iii) & (\doteq) \vdash & \quad \forall v_1 v_2 (\ \ v_1 \doteq v_2 \to \forall x \, (x \in v_1 \leftrightarrow x \in v_2) \ ). \end{array}$$

Then we introduce the *ker-equality* (the reason for such name will be clear in § 3.2 when the concept of *kernel* will be presented in detail) axiom for bags:

$$(E_k^m) \qquad \forall y_1 y_2 v_1 v_2 \left(\begin{array}{c} \{\!\!\{ y_1 \mid v_1 \}\!\!\} \doteq \{\!\!\{ y_2 \mid v_2 \}\!\!\} \ \leftrightarrow \\ (y_1 \doteq y_2 \land v_1 \doteq v_2) \lor \\ \exists z \, (v_1 \doteq \{\!\!\{ y_2 \mid z \}\!\!\} \land v_2 \doteq \{\!\!\{ y_1 \mid z \}\!\!\}) \end{array} \right)$$

which, in a sense, complete properties (i) and (ii) of the previous lemma. Moreover, Lemma 3.16 (iii) states that 'having the same elements' is a necessary conditions for two bags to be considered equal. Axiom (E_k^m) states that two non-empty bags are equal if and only if they have the same number of occurrences of each element. Assuming (DCA) implies that any finite bag is built adding to the empty constant symbol $\{\!\!\{ \ \}\!\!\}$ a finite number of elements using the bag constructor symbol $\{\!\!\{ \cdot \mid \cdot \}\!\!\}$. If (DCA) does not hold, in principle, a bag can be built starting from any other object. Axiom (E_k^m) states, in particular, that a necessary condition for two bags to be equal is that they are based on the same object; we will call such object the *kernel* of the bag.

Assuming (DCA), axiom (E_k^m) forces two bags to be equal if and only if they have the same number of occurrences of each element (eventually none).

In order to state this property in the language itself (for finite sets only), we introduce into Π the denumerable family of predicate symbols $\in^0, \in^1, \in^2, \ldots$, that will be used as shorthands for formulae of the language according to the following definition:

$$\begin{array}{lll} (\in_0) & \forall xy & (x \in^0 y \leftrightarrow x \notin y) \\ (\in_1) & \forall xy & \left(x \in^{n+1} y \leftrightarrow \exists z \, (y \doteq \{\!\!\{ x \mid z \}\!\!\} \land x \in^n z)\right) \end{array}$$

We will refer to the conjunction of all instances of the formulae $(\in_0)$ and $(\in_1)$ as $(\in_*)$. Note that for any x and v, $x \in^n v$ holds in any model of NWE_1 and $(\in_*)$ for exactly one $n \in \omega$.

Lemma 3.17 *For any v_1 and v_2 finite bags, if $NWE_1 \vdash v_1 \doteq v_2$, then $NW \in_* \vdash \forall x\,(x \in^n v_1 \leftrightarrow x \in^n v_2)$.*

Proof. First observe that any model $M = \langle D, I\rangle$ of $NWE_1 \in_*$ is a model of NWE_1, provided $M \models x \in v$ if and only if $M \models x \in^n v$ for some $n > 0$.

Let M be a model of $NWE_1 \in_*$ and assume exist $v_1, v_2, c \in D$ such that $M \models v_1 \doteq v_2$ and $M \models c \in^{n_1} v_1 \wedge c \in^{n_2} v_2$, with $n_1 \neq n_2$. Without loss of generality, assume $n_1 > n_2$. By corollary 3.14, z_1 and z_2 such that $M \models v_1 \doteq \{\!\!\{\underbrace{c,\ldots,c}_{n_2} \,|\, z_1\}\!\!\} \wedge v_2 \doteq \{\!\!\{\underbrace{c,\ldots,c}_{n_2} \,|\, z_2\}\!\!\}$, exist.
By axiom $(\in_*)$, it follows that $c \in^{n_1-n_2} z_1$ and $c \in^0 z_2$; hence $M \models c \in z_1 \wedge c \notin z_2$.
Let $\varphi(x, v)$ be the formula $\exists z\,(v \doteq \{\!\!\{\underbrace{x,\ldots,x}_{n_2} \,|\, z\}\!\!\} \wedge x \in z)$. By axiom $(\doteq_2)$, $M \models \varphi(c, v_1) \to \varphi(c, v_2)$; since $M \models \varphi(c, v_1)$ but $M \models \neg\varphi(c, v_2)$, the claim is true.

3.17 □

The converse is not true (analyze a model based on different kernels). We introduce the *equality principle* for finite multi-sets

$$(E^m) \qquad \forall v_1 v_2 \quad v_1 \doteq v_2 \ \leftrightarrow \text{for all } n\, \forall x\,(x \in^n v_1 \leftrightarrow x \in^n v_2),$$

where n ranges over natural numbers. The finiteness restriction is due to the definition of $(\in_*)$ which is a finite property. We may state the corresponding of Corollary 3.14 for the theory $NW, (\in_*)$, and (E^m). The straightforward proof, basically the definition of $\in^{n+1}$, is omitted.

Lemma 3.18 *If v is a finite bag, then for any $n \geq 0$:*

$$NW \in_* E^m \ \vdash \ \forall x \left(x \in^{n+1} v \leftrightarrow \exists z\,(v \doteq \{\!\!\{ y \,|\, z \}\!\!\} \wedge x \in^n z)\right).$$

We are ready to state

Theorem 3.19 *Let v_1 and v_2 be two finite bags. Then the following are equivalent:*

$$\begin{array}{lll} (i) & NWE_1 & \vdash \ v_1 \doteq v_2, \\ (ii) & NW \in_* E^m & \vdash \ v_1 \doteq v_2, \\ (iii) & NWE_k^m & \vdash \ v_1 \doteq v_2. \end{array}$$

Proof. $(i) \Rightarrow (ii)$. Directly from Lemma 3.17 and the definition of (E^m).

$(ii) \Rightarrow (iii)$. By induction on $\max\{size(v_1), size(v_2)\}$ (the function *size* is defined on ground terms as $size(f(t_1, \dots t_n)) = 1 + \sum_{i=1}^{n} size(t_i)$.
The base case $\{\!\!\{\ \}\!\!\} \doteq \{\!\!\{\ \}\!\!\}$ follows trivially by $(\doteq_1)$.
Let $v_1 \equiv \{\!\!\{ t_1 \,|\, s_1 \}\!\!\}$ and $v_2 \equiv \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}$;

- if $NW \in_* E^m \vdash t_1 \doteq t_2$ then, by induction hypothesis, $NWE^m_k \vdash t_1 \doteq t_2$. In addition, by axioms $(\in_*)$ and (E^m), it is easy to prove that $NW \in_* E^m \vdash s_1 \doteq s_2$. By induction hypothesis and $(\doteq_2)$, $NWE^m_k \vdash \{\!\!\{ t_1 \,|\, s_1 \}\!\!\} \doteq \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}$;

- if $NW \in_* E^m \not\vdash t_1 \doteq t_2$, then let $z_1 \equiv s_1 - \{\!\!\{ t_2 \}\!\!\}$ and $z_2 \equiv s_2 - \{\!\!\{ t_1 \}\!\!\}$ (they exist thanks to Lemma 3.18). By (E^m) it is easy to prove that $NW \in_* E^m \vdash z_1 \doteq z_2$, hence we will simply refer to them as z. By induction hypothesis, $NWE^m_k \vdash s_1 \doteq \{\!\!\{ t_2 \,|\, z \}\!\!\} \wedge s_2 \doteq \{\!\!\{ t_1 \,|\, z \}\!\!\}$. By (E^m_k), $\{\!\!\{ t_1, t_2 \,|\, z \}\!\!\} \doteq \{\!\!\{ t_2, t_1 \,|\, z \}\!\!\}$. Hence $NWE^m_k \vdash \{\!\!\{ t_1 \,|\, s_1 \}\!\!\} \doteq \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}$.

$(iii) \Rightarrow (i)$. Again by induction on $\max\{size(v_1), size(v_2)\}$.
The base case $\{\!\!\{\ \}\!\!\} \doteq \{\!\!\{\ \}\!\!\}$ follows trivially by $(\doteq_1)$.
Let $v_1 \equiv \{\!\!\{ t_1 \,|\, s_1 \}\!\!\}$ and $v_2 \equiv \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}$;

- if $NWE^m_k \vdash t_1 \doteq t_2 \wedge s_1 \doteq s_2$ then, by induction hypothesis and $(\doteq_2)$ $NWE_1 \vdash \{\!\!\{ t_1 \,|\, s_1 \}\!\!\} \doteq \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}$;

- suppose $NWE^m_k \vdash \exists z\,(s_1 \doteq \{\!\!\{ t_2 \,|\, z \}\!\!\} \wedge s_2 \doteq \{\!\!\{ t_1 \,|\, z \}\!\!\})$. Both the equations involve ground terms of *size* fewer than

$$\max\{size(\{\!\!\{ t_1 \,|\, s_1 \}\!\!\}), size(\{\!\!\{ t_2 \,|\, s_2 \}\!\!\})\}\ .$$

  By induction hypothesis $NWE_1 \vdash s_1 \doteq \{\!\!\{ t_2 \,|\, z \}\!\!\}$ and $NWE_1 \vdash s_2 \doteq \{\!\!\{ t_1 \,|\, z \}\!\!\}$. By (E_1), $\{\!\!\{ t_1, t_2 \,|\, z \}\!\!\} \doteq \{\!\!\{ t_2, t_1 \,|\, z \}\!\!\}$.

3.19 □

3.1.3 Compact lists

Let Σ be $\{\llbracket\ \rrbracket, \llbracket \cdot \mid \cdot \rrbracket\}$; the *(left) absorption axiom* (E_2) is defined as follows:

$$(E_2) \qquad \forall xy\, \llbracket x, x \mid y \rrbracket \doteq \llbracket x \mid y \rrbracket$$

The theory obtained by adding (E_2) in NW, briefly denoted as NWE_2 and modeled, for instance, by $H_\Sigma / \equiv_c$ has been scarcely studied. We will give here an intuitive model chosen from real life, which justifies the caption *compact lists.*

Assume a clerk at a pension window hands out forms of two different kinds—`Male`/`Female`—having picked them from their respective stacks.

If two or more consecutive pensioners in the queue require the same form, the wise clerk can fulfill their request with a unique action. There is no great difference[3] for him if the input queue is

$$\llbracket \mathtt{M}, \mathtt{M}, \mathtt{F}, \mathtt{F}, \mathtt{F}, \mathtt{M}, \mathtt{M} \rrbracket \quad \text{or} \quad \llbracket \mathtt{M}, \mathtt{F}, \mathtt{M} \rrbracket.$$

A similar example is the following: assume a laser printer can print on sheets of different size, but has a unique paper feeder. Whenever a new size of the paper is required, an operator should replace sheets in the feeder. Even in this case the input queue can be perfectly modeled by a compact list.

Models for compact lists

Let $\equiv_c$ be a congruence relation on $\tau(\Sigma)$ induced by axiom (E_2) shown above. For each $t \in \tau(\Sigma)$, with $[\, t\,]$ we denote the (canonical) representative of the equivalence class modulo $\equiv_c$ in $\tau(\Sigma)$. A representative can be chosen adopting any judicious criterion; one possibility is the following: *for any representative* $[\, t\,]$ *no subterm of the form* $\llbracket\ x, x \mid z\ \rrbracket$ *occurs in it.* (This is the same thing as require: $[\, t\,]$ *is the minimum membership of the equivalence class with respect to the number of occurrences of the functional symbols* $\llbracket \cdot \mid \cdot\ \rrbracket$ *employed.*)

[3]Clearly this model ceases to hold when the number of consecutive pensioners requiring the same form is larger than the number of forms in the stack. Moreover, even if this is not the case, the clerk cannot ignore the difference of weight between a thousand forms and only one. It is only a model for scarcely accustomed windows.

The Prolog program which returns the canonical representative of each compact list is the following:

```
canon([[ ]], [[ ]]).
canon([[ T | S ]], Can) :-          ev_insert(T, [[ T | R ]], [[ T | R ]]) :-
    canon(T, Tcan),                     !.
    canon(S, Scan),                 ev_insert(T, R, [[ T | R ]]).
    ev_insert(Tcan, Scan, Can).
```

It is easy to see that two hereditarily finite compact-lists ℓ_1 and ℓ_2 are such that $NWE_1 \vdash \ell_1 \doteq \ell_2$ if and only if the goal

:– canon(ℓ_1, Can), canon(ℓ_2, Can)

succeeds (i.e. ℓ_1 and ℓ_2 have the same canonical representative).

Similarly to bags, we are interested in *generated* models of compact lists, namely models whose domains contain only finite terms. In such models, in particular, the following lemmata, which will be useful to define the *occur-check* axiom for bags, hold.

Lemma 3.20 $(E_2) \vdash \forall xy \left(x \doteq \llbracket y \,|\, x \rrbracket \leftrightarrow \exists z \,(x \doteq \llbracket y \,|\, z \rrbracket)\right)$.

Proof. The $(\rightarrow)$ direction follows immediately (pick z as x).
For the $(\leftarrow)$ one, assume $x \doteq \llbracket y \,|\, z \rrbracket$. Then, by (E_2), $x \doteq \llbracket y, y \,|\, z \rrbracket \equiv \llbracket y \,|\, \llbracket y \,|\, z \rrbracket \rrbracket$, i.e. $\llbracket y \,|\, x \rrbracket$.

3.20 □

Lemma 3.21 *In any generated model of NWE_2, for any $y_1, \dots, y_n$, the following holds:*

$\exists x \,(x \doteq \llbracket y_1, \dots, y_n \,|\, x \rrbracket) \leftrightarrow (y_1 \doteq y_2 \doteq \cdots \doteq y_n)$.

Proof. The $(\leftarrow)$ direction follows immediately picking x as $\llbracket y_1 \rrbracket$.
For $(\rightarrow)$, assume $x \doteq \llbracket y_1, \dots, y_n \,|\, x \rrbracket$. Axiom $(\doteq_2)$ implies that

$$\begin{array}{lcl} x & \doteq & \llbracket y_1 \cdots y_n \,|\, x \rrbracket \,, \\ x & \doteq & \llbracket y_1 \cdots y_n, y_1 \cdots y_n \,|\, x \rrbracket \,, \\ x & \doteq & \llbracket y_1 \cdots y_n, y_1 \cdots y_n, y_1 \cdots y_n \,|\, x \rrbracket \,, \\ \vdots & \vdots & \quad\quad \vdots \quad\quad\quad \vdots \end{array}$$

$y_1, \ldots, y_n$ are subdivided into $1 \le k \le n$ classes such that

$$y_1 \doteq \cdots \doteq y_{i_1} \quad \not\doteq \quad y_{i_1+1} \doteq \cdots \doteq y_{i_2} \quad \not\doteq \quad \cdots \quad \not\doteq \quad y_{i_k+1} \doteq \cdots \doteq y_n \, .$$

Whether $y_1 \doteq y_n$ or not, $y_1, \cdots, y_n, y_1, \cdots, y_n$ will be subdivided into $2k$ or into $2k-1$ classes of the form above, respectively.
Similarly, $y_1, \cdots, y_n, y_1, \cdots, y_n, y_1, \cdots, y_n$ will be subdivided into $3k$ or into $3k-2$ classes, and so on.
Since $k \ge 1$, the observation preceding this lemma implies that $y_1 \doteq y_n$ must hold and that $k = 2k-1 = 3k-2 = \cdots$. This (infinite) set of equations has the unique finite solution $k = 1$, i.e. $y_1 \doteq y_2 \doteq \cdots \doteq y_n$.

3.21 □

Observe that the finiteness requirement is necessary for lemmas 3.21. If we were to accept infinite solutions, then, for any $y_1, \ldots, y_n$, the equation $x \doteq [\![\, y_1, \ldots, y_n \,|\, x \,]\!]$ would always admit the infinite solution $x = [\![\, y_1, \ldots, y_n, y_1, \ldots, y_n, y_1, \ldots, y_n, \ldots \,]\!]$.

Equality criterion for compact lists

Similarly to what done for bags, we will define below an equality criterion for compact lists. The equational axiom E_2 cannot state when two objects are distinct: it is consistent for the theory NWE_2 that $[\![\, [\![\;]\!], [\![\, [\![\;]\!] \,]\!] \,]\!] \doteq [\![\, [\![\, [\![\;]\!] \,]\!], [\![\;]\!] \,]\!]$ (it is true in the model $H_\Sigma / \equiv_s$). We will enforce the theory in order to make the data-structure compact-list distinct from the set one. We introduce the following equality principle for compact lists:

$$(E_k^c) \qquad \forall y_1 y_2 v_1 v_2 \left(\begin{array}{l} [\![\, y_1 \,|\, v_1 \,]\!] \doteq [\![\, y_2 \,|\, v_2 \,]\!] \;\; \leftrightarrow \\ \qquad\qquad (y_1 \doteq y_2 \wedge v_1 \doteq v_2) \vee \\ \qquad\qquad (y_1 \doteq y_2 \wedge v_1 \doteq [\![\, y_2 \,|\, v_2 \,]\!]) \vee \\ \qquad\qquad (y_1 \doteq y_2 \wedge [\![\, y_1 \,|\, v_1 \,]\!] \doteq v_2) \end{array} \right)$$

which has its expected meaning for finite terms. The '$\leftarrow$' direction of axiom (E_k^c) makes axiom (F_2) superfluous.

Lemma 3.22 *(E_k^c) implies (E_2)*

Proof. (E_k^c) guarantees that $[\![\, x, x \,|\, y \,]\!] \doteq [\![\, x \,|\, y \,]\!]$ if and only if $x \doteq x \wedge ([\![\, x \,|\, y \,]\!] \doteq y \vee [\![\, x \,|\, y \,]\!] \doteq [\![\, x \,|\, y \,]\!] \vee [\![\, x, x \,|\, y \,]\!] \doteq y)$. $(\doteq_1)$ implies both $x \doteq x$ and $[\![\, x \,|\, y \,]\!] \doteq [\![\, x \,|\, y \,]\!]$ hold.

3.22 □

Hence, the well-founded theory of hereditarily finite compact-lists that we propose extends *NW* with axiom E_k^c and axiom schema F_3^c (occur-check axiom), defined accordingly with the result of Lemmata 3.20 and 3.21:

(F_3^c) $\quad x \not\doteq t[x]$
unless $t[x]$ is of the form $[\![\, t_1, \ldots, t_n \,|\, x \,]\!]$,
and $t_1 \doteq \cdots \doteq t_n$

where, $t[x]$ denotes a term having x as proper subterm.

Non-well-founded (even infinite) theories of compact lists are described in an extended context in § 3.2.3.

3.1.4 Sets

Let $\mathcal{L}$ be such that $\Pi = \{\doteq, \in\}$ and $\Sigma = \{\emptyset, \{\,\cdot\,|\,\cdot\}\,\}$. The theory NWE_1E_2 written in the language $\mathcal{L}(\Pi, \Sigma)$ will be the starting point for the analysis of the set theories useful in Logic Programming:

(E_1) $\quad \forall xyz\, \{x, y \,|\, z\} \doteq \{y, x \,|\, z\}$
(E_2) $\quad \forall xz\, \{x, x \,|\, z\} \doteq \{x \,|\, z\}$

The first presentation of a set theory based on the simple constructors $\emptyset$ and $\{\,\cdot\,|\,\cdot\}$ (the `with` constructor) can be found in [17].

In the rest of this section we first describe in detail a privileged model for NWE_1E_2, and then we will compare different equality criteria (extensionality axioms) for sets.

Models for sets

Theorem 3.4 ensures that any model of *NW* (hence of NWE_1E_2) must necessarily be infinite. Furthermore, it is easy to see that any model of NWE_1E_2 must contain an isomorphical copy of the model $H_\Sigma/\equiv_s$ of the hereditarily finite and well-founded sets. $H_\Sigma/\equiv_s$ reflects the intuitive semantics of sets; moreover, in this model, all regularity axioms presented in § A.2 become equivalent, as well as all equality criteria presented below. Moreover, $H_\Sigma/\equiv_s$ is the *initial model* of the equational theory $(E_1)(E_2)$.

We repeat here its definition. Let $\equiv_s$ be a congruence relation on $\tau(\Sigma)$ determined by axioms (E_1) and (E_2). For each $t \in \tau(\Sigma)$, with $[\,t\,]$ we denote the (canonical) representative of the equivalence class modulo $\equiv$ in $\tau(\Sigma)$. With $H_\Sigma/\equiv_s$ we denote the structure $\langle\{[\,t\,] : t \in \tau(\Sigma)\}, I\rangle$, where $t^I = [t]_{\equiv_s}$, for any term t, $t_1 \doteq^I t_2$ is `true` if and only if $[t_1]_{\equiv_s} = [t_2]_{\equiv}$, and $t_1 \in^I t_2$ is `true` if and only if $[t_2]_{\equiv_s}$ is of the form $\{\cdots, [t_1]_{\equiv_s}, \cdots\}$.

The following Prolog program returns the canonical representative of any ground term t.

```
canon(∅, ∅).
canon({T | S}, Can) :−
    canon(T, Tcan),
    canon(S, Scan),
    ev_insert(Tcan, Scan, Can).

ev_insert(T, ∅, {T}).
ev_insert(T, {T | R}, {T | R}) : −
    !.
ev_insert(T, {S | R}, {T, S | R}) :−
    get_value(T, ValT),
    get_value(S, ValS),
    ValS < ValT,
    !.
ev_insert(T, {S | R1}, {S | R2}) :−
    ev_insert(T, R1, R2).

get_value(∅, 0).
get_value({T | S}, Val) :−
    get_value(T, V1),
    get_value(S, V2),
    exp(2, V1, Val1),
    Val is Val1 + V2.
```

Note that $\emptyset$ and $\{\cdot\,|\,\cdot\}$ are here used free (uninterpreted) and they can be replaced by standard $[\,]$ and $[\cdot\,|\,\cdot]$, respectively. exp(A, B, C) returns in C the value A^{B}.

Consider the ordering $\overset{\mathcal{O}}{\prec}$ of $\{[\,t\,] : t \in \tau(\Sigma)\}$, defined as follows:

- for any x and y, $\emptyset \overset{\mathcal{O}}{\prec} \{x \,|\, y\}$;
- Let $x = \{s_0, \ldots, s_m\}$ and $y = \{t_0, \ldots, t_n\}$ be such that $t_i \overset{\mathcal{O}}{\prec} t_j$ and $s_i \overset{\mathcal{O}}{\prec} s_j$ for $i > j$ (the lighter elements will take the larger index); then

$$x \overset{\mathcal{O}}{\prec} y \quad \text{if and only if} \quad s_0 \overset{\mathcal{O}}{\prec} t_0 \vee \{s_1, \ldots, s_m\} \overset{\mathcal{O}}{\prec} \{t_1, \ldots, t_n\}$$

We recall from the literature (see e.g. [70]) the *Ackermann ordering.* Bind any hereditarily finite x to an integer n as follows (here $\{s_1, \ldots, s_n\}$ is the real *set*):

$$\begin{cases} ack(\emptyset) & = & 0 \\ ack(\{x_1, \ldots, x_n\}) & = & \sum_{i=1}^{n} 2^{ack(x_i)} \end{cases}$$

Then $x \overset{\mathcal{A}}{\prec} y$ if and only if $ack(x) < ack(y)$.

Theorem 3.23 *Let x and y be hereditarily finite and well-founded sets. Then $x \overset{\mathcal{O}}{\prec} y$ if and only if $x \overset{\mathcal{A}}{\prec} y$.*

Proof. It follows immediately from the inequality $2^{n+1} > \sum_{i=0}^{n} 2^i$, that we prove by mathematical induction on n.
Base) $(n = 0)$: $2^1 = 2 > 2^0 = 1$.
Step) Assume (induction hypothesis) that $2^n > \sum_{i=0}^{n-1} 2^i$. This means that $2^n \geq 1 + \sum_{i=0}^{n-1} 2^i$. Hence $2^{n+1} \geq 2 + \sum_{i=0}^{n-1} 2^{i+1} = 1 + \sum_{i=0}^{n} 2^i > \sum_{i=0}^{n} 2^i$.

3.23 □

It is easy to see that the goal

:− $\mathsf{get\_value}(\mathsf{S}, \mathsf{V})$

returns in V exactly the value $ack(\mathsf{S})$.

A non-well-founded universe

We will describe now a non-well-founded model of the theory NWE_1E_2 of which $H_\Sigma/\equiv_s$ is a submodel.

All elements of the model $H_\Sigma/\equiv_s$ are representable by suitable terms (cf. Def. 2.2) having finite domain. This abstract view of ground terms becomes almost mandatory when one comes to consider the generalized kind of terms that form the *completion* of a Herbrand universe: typically this is done by withdrawing the requirement that labeled trees must have finitely many nodes.[4] From this graph-theoretical perspective, syntactic equivalence between terms turns out to coincide with the notion of isomorphism between labeled, ordered trees (cf. § 2.1).

Adjusting the complete Herbrand universe definition to the case when the construction of the universe is not entirely free, due to the presence of the set constructor symbol in the signature one obtain a universe of non-well-founded sets or *hyperset* [13].

The intuitive semantics of this construct must reflect into the criteria we adopt for finding equivalences between rooted graphs. Such criteria cease accordingly, in our specialized context, to be purely syntactic. At an even more fundamental level, we will have to discard certain trees labeled over Σ, that cannot be regarded as ground terms due to the semantics of $\{\cdot\,|\,\cdot\}$.

Discarding axiom (DCA), we may label nodes with symbols distinct from $\emptyset$ and $\{\cdot\,|\,\cdot\}$. This provides a definition of model more general than the one needed here; however, it will be useful to model the hybrid theory of sets described in § 3.2.4.

The terms whose root bears a label distinct from $\{\cdot\,|\,\cdot\}$ will be regarded as memberless entities, named *colors*: we will allow insertions like $\{X\,|\,c\}$ for any color c distinct from $\emptyset$, regarding any hyperset that results from an insertion of this kind as something distinct from $\{X\}$.

To avoid under-specified situations, we add to the definition of ground term (cf. Def. 2.2) the following:

[4]Alternatively, one could characterize generalized ground terms by means of systems, possibly infinite, of equations involving ordinary non ground terms (cf. [30, 78]).

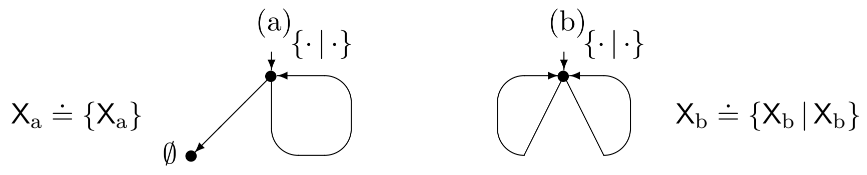

(a)
$\{\cdot\,|\,\cdot\}$
$\mathsf{X_a} \doteq \{\mathsf{X_a}\}$
$\emptyset$
(b)
$\{\cdot\,|\,\cdot\}$
$\mathsf{X_b} \doteq \{\mathsf{X_b}\,|\,\mathsf{X_b}\}$

Figure 3.2: Two terms that cannot be regarded as ground.

> **Groundness restriction.** *The requirement henceforth becomes integral part of the definition of (ground)* term *that there be no infinite sequence $\nu_0, \nu_1, \nu_2, \ldots$ of nodes with $\mathcal{T}(\nu_i) = \{\cdot\,|\,\cdot\}$ and $\nu_{i+1} = [\nu_i, 2]$ for all i.*

To see why the presence of a path $\nu_0, \nu_1, \nu_2, \ldots$ as above, in a term, would conflict with the very notion of groundness, let us examine the two graphs of Fig. 3.2.

Either of them is the picture of a labeled tree that violates the groundness restriction. The left arc in either graph only indicates self-inclusion; hence it conveys no information about the entity ($\mathsf{X_a}$ and $\mathsf{X_b}$ respectively) represented by the root. The first arc of Fig. 3.2(a) indicates that $\emptyset$ must belong to $\mathsf{X_a}$, a property which is clearly insufficient to characterize $\mathsf{X_a}$. The right arc of Fig. 3.2(b) indicates that $\mathsf{X_b}$ must belong to itself. If $\mathsf{X_b}$ were to be an ordinary set, this would be an absurdity, but we are dealing with hypersets here. Since membership $\underline{\text{can}}$ form cycles among such entities, we are again facing an under-specified situation.

Our next step will be to get rid of irrational terms (an interesting example of set-theoretic irrational term is the one whose picture is the graph of Fig. 3.3). Preliminary to that, we need the notion of *bisimulation*, which in turn presupposes the following couple of auxiliary notions.
For every term $\mathcal{T}$ and any ν in $dom(\mathcal{T})$, let $\tau_0, \ldots, \tau_g$ and $\mu_0, \ldots, \mu_{g-1}$ be the sequences of nodes such that:

- $\tau_0 = \nu$;
- $\mathcal{T}(\tau_i) = \{\cdot\,|\,\cdot\}$, $\mu_i = [\tau_i, 1]$ and $\tau_{i+1} = [\tau_i, 2]$ for $i = 0, \ldots, g-1$;

- $\mathcal{T}(\tau_g) \neq \{\cdot \mid \cdot\}$.

We denote by $Color(\nu)$ the node τ_g and call $\in$-*predecessors* of ν the μ_is.

Definition 3.24 *Let $\mathcal{T}_0, \mathcal{T}_1$ be terms. A relation*

$$\mathcal{B} \subseteq dom(\mathcal{T}_0) \times dom(\mathcal{T}_1)$$

is said to be a Bisimulation *between $\mathcal{T}_0$ and $\mathcal{T}_1$ if and only if:*

i) $[\,]\,\mathcal{B}\,[\,]$,

ii) *when $\nu_0\,\mathcal{B}\,\nu_1$, the following hold:*

- *$Color_0(\nu_0)\,\mathcal{B}\,Color_1(\nu_1)$, $\mathcal{T}_0(\nu_0) = \mathcal{T}_1(\nu_1)$, and moreover*
- *to every $\in$-predecessor ϱ_b of ν_b in $\mathcal{T}_b$ ($b = 0$ or $b = 1$), there corresponds at least one $\in$-predecessor ϱ_{1-b} of ν_{1-b} in $\mathcal{T}_{1-b}$ such that $\varrho_0\,\mathcal{B}\,\varrho_1$;*
- *if $\mathcal{T}_0(\nu_0) \neq \{\cdot \mid \cdot\}$, then $[\nu_0, i]\mathcal{B}[\nu_1, i]$ for $i = 1, \ldots, ar(\mathcal{T}_0(\nu_0))$.*

We write $\mathcal{T}_0 \approx \mathcal{T}_1$ if and only if there is a bisimulation $\mathcal{B}$ between $\mathcal{T}_0$ and $\mathcal{T}_1$.

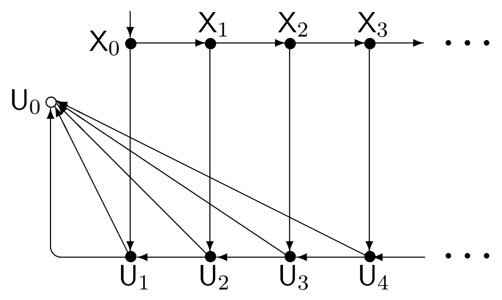
Bisimulations are, in a sense, isomorphisms complying with the intended (hyperset) semantics of $\{\cdot \mid \cdot\}$. Accordingly, $\mathcal{T}$ will be regarded as a *rational* term if and only if it has only finitely many subterms that cannot bisimulate one another. To make this idea precise, let us denote by $\mathcal{T}\lceil\nu$ the subterm of $\mathcal{T}$ issuing from a given node ν.

Definition 3.25 *A ground term $\mathcal{T}$ is said to be* Rational *if and only if there are $\nu_0, \ldots, \nu_m$ in $dom(\mathcal{T})$, with m in ω, such that for every μ in $dom(\mathcal{T})$ there is an i, $0 \leq i \leq m$, fulfilling $\mathcal{T}\lceil\nu_i \approx \mathcal{T}\lceil\mu$.*

In conclusion,

Definition 3.26 *Indicating by $\overline{\mathbb{G}}_\Sigma$ the family of all rational terms over Σ, our* Hyperset Universe *will be*

$$\overline{\mathbb{H}}_\Sigma = \overline{\mathbb{G}}_\Sigma / \approx \ .$$



Legenda:

$\mathcal{T}(\bullet) = \{\cdot \mid \cdot\}$
$\mathcal{T}(\circ) = \emptyset$

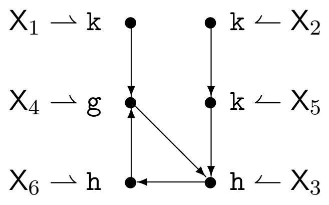
$X_i \doteq \{X_{i+1}, U_i\}$
$U_0 \doteq \emptyset$
$U_{i+1} \doteq \{U_i\} (\, i = 0, 1, 2, \ldots)$

Figure 3.3: A cycle-free irrational hyperset

$X_1 \doteq \mathtt{k}(\mathtt{g}(X_3))$

$X_2 \doteq \mathtt{k}(\mathtt{k}(X_3))$

$X_3 \doteq \mathtt{h}(\mathtt{h}(\mathtt{g}(X_3)))$



$X_1 \rightharpoonup \mathtt{k}$
$X_4 \rightharpoonup \mathtt{g}$
$X_6 \rightharpoonup \mathtt{h}$
$\mathtt{k} \leftharpoonup X_2$
$\mathtt{k} \leftharpoonup X_5$
$\mathtt{h} \leftharpoonup X_3$

$X_1 \doteq \mathtt{k}(X_4)$	$X_2 \doteq \mathtt{k}(X_5)$
$X_3 \doteq \mathtt{h}(X_6)$	$X_4 \doteq \mathtt{g}(X_3)$
$X_5 \doteq \mathtt{k}(X_3)$	$X_6 \doteq \mathtt{h}(X_4)$

Figure 3.4: Two renderings (one of them 'flat') of the same graph.

Let us now broaden the discourse by adjoining to our former signature Σ the denumerably infinite collection $\mathcal{V}$ of *variables*. Labeled graphs, and in particular terms, whose labeling may involve variables, or that may violate the above-stated groundness restriction, will be said to be *hollow*. As will emerge from § 4.3.3, every hollow, rooted and finite graph depicts a collection of ground terms, obtainable from it via substitutions.

As illustrated by Fig. 3.3 and 3.4, any ground labeled graph $\mathcal{G}$ (possibly with cycles) can be variously *rendered*, up to isomorphism, as a conjunction (singleton when $\mathcal{G}$ is finite and acyclic; infinite when $\mathcal{G}$ is not rational)

$$\bigwedge_{\nu \text{ in } \mathcal{C}} (X_\nu \doteq t_\nu)$$

of first-order equalities over the signature $\Sigma \cup \mathcal{V}$, where

- $\mathcal{C}$ is a collection of nodes of $\mathcal{G}$ comprising all nodes of $\mathcal{G}$ devoid of entering arcs, along with at least one node lying on ϖ for each infinite path ϖ of $\mathcal{G}$;

- the X_ν's belong to $\mathcal{V}$, hence they are not used as labels in $\mathcal{G}$, and they are distinct from one another; the t_ν's are first-order terms[5] over the signature $\Sigma \cup \{X_\nu \,:\, \nu \text{ in } \mathcal{C}\}$.

To do that, one views $\mathcal{G}$ as a collection $\{\, \mathcal{G}_\nu \,:\, \nu \,\text{in}\, \mathcal{C} \,\}$ of finite acyclic rooted graphs labeled over $\Sigma \cup \{\, X_\nu \,:\, \nu \text{ in } \mathcal{C} \,\}$, each ν in $\mathcal{C}$ bearing the label X_ν and each $\mathcal{G}_\nu$ being in a sense 'grafted' into ν; then one takes as t_ν the first-order term that straightforwardly corresponds to $\mathcal{G}_\nu$.[6]

Viewed this way, a rational ground labeled graph $\mathcal{G}$ is just a special case of what is usually called a *Herbrand system* (cf. Def. 2.12).

An axiomatization for our hypersets will be proposed in § 3.2.4.

Equality criteria for sets

In the remaining part of this section we want to point out the relations between three different forms of the equality criteria for sets: the classical one (extensionality axiom—see e.g. [64])

$$(E) \qquad x \doteq y \leftrightarrow \forall z \, (z \in x \leftrightarrow z \in y) \, ;$$

the combination of axioms (E_1) and (E_2)

$$\begin{array}{lll} (E_1) & \forall xyz & (\{x, y \,|\, z\} \;\doteq\; \{y, x \,|\, z\}) \\ (E_2) & \forall xz & (\{x, x \,|\, z\} \;\doteq\; \{x \,|\, z\}) \, , \end{array}$$

and, lastly, based on the same stream of ideas that guided the introduction of axioms (E_k^m) and (E_k^c), the axiom (E_k^s), very useful in practice:

$$\begin{array}{ll} (E_k^s) & \{x \,|\, v\} \doteq \{y \,|\, w\} \leftrightarrow \\ & \quad (x \doteq y \wedge v \doteq w) \vee \\ & \quad (x \doteq y \wedge v \doteq \{y \,|\, w\}) \vee \\ & \quad (x \doteq y \wedge w \doteq \{x \,|\, v\}) \vee \\ & \exists z \, (v \doteq \{y \,|\, z\} \wedge w \doteq \{x \,|\, z\}) \, . \end{array}$$

[5]Notice the distinction we are making between first-order (concrete) terms and terms in the graph-theoretical sense.

[6]When $\mathcal{G}$ is rooted, finite and acyclic, so that its rendering is $X \doteq t$, X not occurring in t, t itself is called *rendering* of $\mathcal{G}$.

We will use axiom (DCA), namely $\forall x\left(x \doteq \emptyset \vee \exists yz\,(x \doteq \{y\,|\,z\})\right)$, to avoid to deal with uncontrolled functional symbols. As usual, we denote $x \subseteq y$ for the formula $\forall z(z \in x \rightarrow z \in y)$.

The two following facts trivially hold:

1. $E \vdash x \doteq y \leftrightarrow (x \subseteq y) \wedge (y \subseteq x)$;

2. $W \vdash \{w_1\,|\,v_1\} \subseteq \{w_2\,|\,v_2\} \leftrightarrow \Big((w_1 \doteq w_2 \vee w_1 \in v_2) \wedge v_1 \subseteq v_2\Big)$.

Theorem 3.27 *In any model of NW and (DCA) in which every object of the domain contains finite (possibly 0) elements, then (E) implies (E^s_k).*

Proof. It is easy to see that

$$\begin{array}{l}
E(DCA) \vdash \forall xy\,(x \doteq y \leftrightarrow \\
\qquad (x \doteq \emptyset \wedge y \doteq \emptyset \wedge x \subseteq y \wedge y \subseteq x)\vee \\
\qquad (x \doteq \emptyset \wedge \exists v_2 w_2\, y \doteq \{w_2\,|\,v_2\} \wedge x \subseteq y \wedge y \subseteq x)\vee \\
\quad (\exists v_1 w_1\, x \doteq \{w_1\,|\,v_1\} \wedge y \doteq \emptyset \wedge x \subseteq y \wedge y \subseteq x)\vee \\
\quad (\exists v_1 w_1\, x \doteq \{w_1\,|\,v_1\} \wedge \exists v_2 w_2\, y \doteq \{w_2\,|\,v_2\} \wedge x \subseteq y \wedge y \subseteq x)\ ).
\end{array}$$

Axiom (N) forces the first disjunct to be equivalent to $x \doteq \emptyset \wedge y \doteq \emptyset$ and forces the second and third to fail.

Thanks to the property (2), stated just above the claim of this Theorem, the fourth disjunct simplifies to

$$\begin{array}{l}
\exists v_1 w_1 v_2 w_2\, x \doteq \{w_1\,|\,v_1\} \wedge y \doteq \{w_2\,|\,v_2\}\wedge \\
(w_1 \doteq w_2 \vee (w_1 \in v_2 \wedge w_2 \in v_1)) \wedge v_1 \subseteq y \wedge v_2 \subseteq x))
\end{array}$$

which can be opened in two disjuncts.

- The first, $\psi_1 \equiv x \doteq \{w_1\,|\,v_1\} \wedge y \doteq \{w_2\,|\,v_2\} \wedge w_1 \doteq w_2 \wedge v_1 \subseteq y \wedge v_2 \subseteq x$. We sometimes refer to both w_1 and w_2 as w_1 (they must be equal for satisfying ψ_1). Since

  $$\begin{array}{rcl}
  WE & \vdash & (v_1 \subseteq \{w_1\,|\,v_2\} \wedge v_2 \subseteq \{w_1\,|\,v_1\}) \rightarrow \\
  & & (v_1 \doteq v_2 \vee v_1 \doteq \{w_1\,|\,v_2\} \vee v_2 \doteq \{w_1\,|\,v_1\})\,,
  \end{array}$$

  then ψ_1 implies $(w_1 \doteq w_2) \wedge (v_1 \doteq v_2 \vee v_1 \doteq \{w_2\,|\,v_2\} \vee v_2 \doteq \{w_1\,|\,v_2\})$.

- The second disjunct will be $\psi_2 \equiv x \doteq \{w_1 \,|\, v_1\} \wedge y \doteq \{w_2 \,|\, v_2\} \wedge w_1 \in v_2 \wedge w_2 \in v_1 \wedge v_1 \subseteq y \wedge v_2 \subseteq x$. Let $h_1 = v_1 \setminus \{w_1, w_2\}$ and $h_2 = v_2 \setminus \{w_1, w_2\}$ (their existence is ensured by the finiteness requirement of the model).

  We first show that $WE \vdash h_1 \doteq h_2$; let $z \in h_1$; then $z \in v_1 \wedge z \neq w_1 \wedge z \neq w_2$. Since $v_1 \subseteq y$, this means that $z \in \{w_2 \,|\, v_2\} \wedge z \neq w_1 \wedge z \neq w_2$. By definition of h_2, $z \in h_2$. For the inclusion $h_2 \subseteq h_1$ the proof is similar. We refer to both h_1 and h_2 as h_1.

  (E) forces $v_1 \doteq \{w_2 \,|\, h_1\} \wedge v_2 \doteq \{w_1 \,|\, h_1\}$.

We have proved that

$$\begin{array}{lll} \forall xy & (x \doteq y \rightarrow & (x \doteq \emptyset \wedge y \doteq \emptyset) \vee \\ & & (\exists v_1 w_1 v_2 w_2 \; x \doteq \{w_1 \,|\, v_1\} \wedge y \doteq \{w_2 \,|\, v_2\} \wedge \\ & & \qquad (w_1 \doteq w_2 \wedge v_1 \doteq v_2) \vee \\ & & \qquad (w_1 \doteq w_2 \wedge v_1 \doteq \{w_2 \,|\, v_2\}) \vee \\ & & \qquad (w_1 \doteq w_2 \wedge v_2 \doteq \{w_1 \,|\, v_1\}) \vee \\ & & \quad \exists h_1 \, (v_1 \doteq \{w_2 \,|\, h_1\} \wedge v_2 \doteq \{w_1 \,|\, h_1\})) \end{array}$$

The $\leftarrow$ direction follows immediately.

3.27 □

Remark 3.28 *We are here interested in interpreting finite aggregates only. If one wishes to extend the result of Theorem 3.27 for infinite sets, then it is sufficient to insert in the theory the* Removal Axiom

$$(L) \qquad \forall yv \, \exists \ell \Big(\forall x \, (x \in \ell \leftrightarrow x \in v \wedge x \neq y) \Big) \, ,$$

which, once skolemized, takes the form

$$(L) \qquad \forall xyv \Big(x \in v \; \texttt{less} \; y \leftrightarrow x \in v \wedge x \neq y \Big) \, .$$

Remark 3.29 *(DCA) is necessarily needed to prove Theorem 3.27. Assume that $M = \langle D, I \rangle$ is a model of NW, and assume that $\bar{\emptyset} = \emptyset^I$ and $\bar{c} \in D$ is also such that $\forall x \, (x \notin \bar{c})$. Then $\{\bar{\emptyset} \,|\, \bar{\emptyset}\}$ and $\{\bar{\emptyset} \,|\, \bar{c}\}$ are extensionally equal. However they are not equal neither for $(E_1)(E_2)$ nor for (E^s_k).*

Remark 3.30 *We will define a model $M = \langle D, I\rangle$ of $NWE_1E_2(DCA)$ in which there is an object v containing infinite elements and such that the formula introducing the less operation does not hold for it.*

Let HF be the domain of the model $H_\Sigma/\equiv_s$ defined in § 3.1 (namely, the set of all hereditarily finite and well-founded sets). The domain D is the smallests set such that:

- *$HF \subseteq D$;*
- *the elements $v, v_1, v_2, \ldots$ fulfilling the properties:*

  $$\begin{array}{rcl} v &=& \{\emptyset \mid v_1\} \\ v_1 &=& \{\{\emptyset\} \mid v_2\} \\ v_2 &=& \{\{\{\emptyset\}\} \mid v_3\} \\ \vdots & \vdots & \vdots \\ \{\emptyset, \{\emptyset\}\} &\in& v \end{array}$$

  belongs to D.
- *close $D \cup \{v, v_1, v_2, \ldots\}$ to correctly model the (W) axiom.*

The set v `less` $\{\emptyset, \{\emptyset\}\}$ does not belong to D.

Theorem 3.31 *In any model of (W), axiom (E) implies both axiom (E_1) and axiom (E_2).*

Proof. Axiom (W) ensures that the two sets $\{x, y \mid z\}$ and $\{y, x \mid z\}$ contains exactly the same elements. The same holds for $\{x \mid z\}$ and $\{x, x \mid z\}$.

3.31 □

Theorem 3.32 *(E^s_k) implies (E_1) and (E_2).*

Proof. Applying the disjunction opened by (E^s_k) we obtain in particular:

$$\begin{array}{lll}(E^s_k) & \vdash & \{x, y \,|\, z\} \doteq \{y, x \,|\, z\} \leftrightarrow \\ & & \ldots \vee \underbrace{\exists v\,(\{y \,|\, z\} \doteq \{y \,|\, v\} \wedge \{x \,|\, z\} \doteq \{x \,|\, v\})}_{\text{pick } v = z.\ \text{True by } (\doteq_1)} \vee \ldots \\ (E^s_k) & \vdash & \{x \,|\, z\} \doteq \{x, x \,|\, z\} \leftrightarrow \\ & & \ldots \vee \underbrace{x \doteq x \wedge \{x \,|\, z\} \doteq \{x \,|\, z\}}_{\text{True by } (\doteq_1)} \vee \ldots\end{array}$$

3.32 □

Theorem 3.33 *In any model of NW and (DCA) in which every object of the domain contains finite (possibly* 0*) elements, then* (E^s_k) *implies* (E) *and axioms* (E_1) *and* (E_2) *imply axiom* (E).

Proof. Let x and y be non-empty sets. Assume that they are not extensionally equal; this means that there exists $z \in x$ such that $z \in y$ (or, symmetrically, $z \in y$ and $z \in x$). The finiteness requirement, ensures the existence of a set w_1 such that $x \doteq \{z \,|\, w_1\}$. *(DCA)* ensures that, since y is not empty, exist v_2 and w_2 such that $y \doteq \{v_2 \,|\, w_2\}$. Since $z \notin y$, axiom (W) states that $z \not\doteq v_2$ and $z \notin w_2$. Hence, x and y cannot be considered equal either by (E^s_k) or by $(E_1)(E_2)$.

3.33 □

Theorem 3.34 *In any model of* (E_1) *and* (E_2) *in which all elements are finite, then axiom* (E^s_k) *holds.*

Proof. Operational axiom (E^s_k) contains inside it (in the fourth disjunct) the (L) axiom, which is a theorem for finite sets. This is the only difference between the two axioms, since the first three disjuncts reflects precisely axiom (E_2) while the fourth is the counter-part of axiom (E_1).

3.34 □

3.2 Hybrid theories

In this section we will extend the signature Σ containing the constant symbol $[\,]$ ($\{\!\{\ \}\!\}$, $[\![\ ]\!]$, $\emptyset$) and the binary symbol $[\cdot \,|\, \cdot]$ ($\{\!\{ \cdot \,|\, \cdot \}\!\}$, $[\![\cdot \,|\, \cdot]\!]$,

$\{\cdot\,|\,\cdot\}$), able to describe pure list (bag, compact list, and set) theory, as shown in § 3.1, with a (finite or infinite) number of functional symbols, together with their arities. Such functional symbols will take the role of the standard *free* functional symbols mainly used in Logic Programming. Possible terms are, therefore, standard terms, lists (bags, compact lists, and sets) of standard terms, and any possible combination among them.

In the context of lists, as well as in the other axiomatic theories we will consider, objects denoted by *ground* terms are forced to have a *finite* number of elements. Moreover, it is easily seen that in every model of each of the considered theories in which all elements are denoted by finite terms (we will briefly denote such models as GENERATED MODELS, the membership cannot form either cycles or infinite descending chains. To this extent such theories can be considered well-founded. We will first consider this case; then we will extend the axiomatization to non-well-founded and finite or even non-well-founded and rational theories.

The axiomatizations presented can easily be combined in order to obtain axiomatic theories capable to deal with any subset of the collection of proposed data-structures. Moreover, the unification algorithms that will be presented in Chapter 4 can easily be merged to solve the unification problem relative to such 'combined' context.

As in § 3.1, the set Π of predicate symbols consists of '$\doteq$' and '$\in$'. Axioms ($\doteq_1$), ($\doteq_2$), (N), and (W) will always hold (as shown in Theorem 3.4, this is sufficient to force any model to be infinite).

The Kernel Problem

Assume that the constant symbol 'a' belongs to the signature Σ; a term t of the form $[t_1, \ldots, t_n \,|\, a]$ can be written. (W) states that $t_1, \ldots, t_n$ are elements of t. Remaining elements of t are those of a.

To keep the notion of list (bag, compact list, set) distinct from the notion of 'free' term, we will enforce axiom (N) so as to keep 'empty' any term which is not a list of terms.

Therefore we introduce the axiom schema

$$(K) \qquad \forall x\, y_1 \cdots y_n\, (x \notin f(y_1, \ldots, y_n))$$

for any $f \in \Sigma$, f distinct from $[\,\cdot\,|\,\cdot\,]$ ($\{\!\!\{\,\cdot\,|\,\cdot\,\}\!\!\}$, $[\![\,\cdot\,|\,\cdot\,]\!]$, $\{\,\cdot\,|\,\cdot\,\}$), $ar(f) = n$. ('K' stands for *kernel*.) Observe that—since $[\,]$ belongs to Σ—(N) is an instance of (K).

Definition 3.35 *We will denote as* Ur-Element *(cf. [58]) any term of the form $f(t_1, \dots, t_n)$, where $f \in \Sigma$, f distinct from $[\,\cdot\,|\,\cdot\,]$ ($\{\!\!\{\,\cdot\,|\,\cdot\,\}\!\!\}$, $[\![\,\cdot\,|\,\cdot\,]\!]$, $\{\,\cdot\,|\,\cdot\,\}$), $ar(f) = n$. When an ur-element is ground, it is called a* Kernel *or a* Seed*.*

From now on, if the contrary is not explicitly stated, with f and g we will denote the main functional symbols of an ur-element.

Since we are particularly interested to axiomatize objects denoted by terms, when we deal with the well-founded case we will make a Meta-use of a function, the *kernel extraction function ker* defined as follows:

$$\begin{cases} ker(f(t_1, \dots, t_n)) = f(t_1, \dots, t_n) \\ ker([t_1 \,|\, t_2]) = ker(t_2) \qquad \text{also for } \{\!\!\{\cdot\,|\,\cdot\}\!\!\},\ [\![\cdot\,|\,\cdot]\!], \text{ and } \{\cdot\,|\,\cdot\} \end{cases}$$

In this way we can extend the axioms (E^m) and (E^s), presented for the pure case, to the hybrid case.

Being an element-less term, $\text{ker}(t)$ can easily be proved to be distinct from any list term $[t \,|\, s]$. Nevertheless, nothing is stated by NW about the relations between two ur-elements $f(s_1, \dots, s_m)$ and $g(t_1, \dots, t_n)$.

A possible model $\langle D, I \rangle$ can be obtained mapping all constants of Σ into $[\,]^I$, and, for any $f \in \Sigma$, f distinct from $[\,\cdot\,|\,\cdot\,]$, $ar(f) = n$, putting $f^I([\,]^I, \dots, [\,]^I) = [\,]^I$. Nevertheless, this interpretation does not allow to take advantage of such enrichment of Σ. A reasonable model is the Herbrand model or the complete Herbrand model (cf. Def. 2.4).

When also infinite objects are allowed, the function *ker*, inductively defined to return the kernel of a list (bag, compact list, set) ceases to work. An axiomatization able to control the behavior of kernel entities also for infinite objects is required. In this a meta-use of *ker* is impossible: the signature must be extended in order to contain *ker*. Luckily it is possible to develop the theories we need without explicitly adding the following axioms in the theory:

$$
\begin{array}{ll}
(\mathit{ker}_0) & \mathit{ker}(f(t_1,\dots,t_n)) \doteq f(t_1,\dots,t_n) \\
 & \quad (f \in \Sigma \setminus \{\,[\cdot\,|\,\cdot], \{\!\!\{\cdot\,|\,\cdot\}\!\!\}, [\![\cdot\,|\,\cdot]\!], \{\cdot\,|\,\cdot\}\,\}) \\
(\mathit{ker}_1) & \mathit{ker}([t_1\,|\,t_2]) \doteq \mathit{ker}(t_2) \\
 & \quad (\text{The same for } \{\!\!\{\cdot\,|\,\cdot\}\!\!\},\ [\![\cdot\,|\,\cdot]\!], \text{ and } \{\cdot\,|\,\cdot\})
\end{array}
$$

As explained in Remarks 3.15 and 3.30, when infinite objects are considered, a removal axiom (L) is needed to guarantee useful properties for bags and sets, such as, for instance:

$$
\begin{array}{ll}
\texttt{Bags} \vdash & y \in x \to \exists z\,(x \doteq \{\!\!\{\,y\,|\,z\,\}\!\!\}) \\
\texttt{Sets} \vdash & y \in x \to \exists z\,(x \doteq \{y\,|\,z\} \wedge y \notin z)
\end{array}
$$

In this case we need to add a further axiom concerning with the removal operator `less` defined in Remark 3.28.

$$(\mathit{ker}_2) \qquad \mathit{ker}(s\ \texttt{less}\ t) \quad \doteq \quad \mathit{ker}(s)$$

Lastly, we need to be sure that a kernel entity does not contain elements. This can be done introducing axiom (K'):

$$(K') \qquad \forall xy\,(x \notin \mathit{ker}(y))$$

or, alternatively, introducing a sort of (DCA) axiom for kernels which, together with axiom (K), implies (K'):

$$(DCA_{\mathit{ker}}) \quad \forall x \exists y_1 \dots y_A \bigvee_{f \in \Sigma \setminus \{[\cdot|\cdot], \{\!\{\cdot|\cdot\}\!\}, [\![\cdot|\cdot]\!], \{\cdot|\cdot\}\}} \mathit{ker}(x) \doteq f(y_1,\dots,y_{ar(f)})$$

where $A = \mathit{max}\{ar(f) : f \in \Sigma \setminus \{[\cdot\,|\,\cdot], \{\!\!\{\cdot\,|\,\cdot\}\!\!\}, [\![\cdot\,|\,\cdot]\!], \{\cdot\,|\,\cdot\}\}\,\}$.

Axiomatization of the Herbrand universe

In [28] Clark provides an axiomatization (shown to be complete in [74]) for a signature Σ when Π is simply $\{\doteq\}$. Such axiomatization can be summarized by the three axiom schemata

$$
\begin{array}{llll}
(F_1) & \forall x_1 \cdots x_n y_1 \cdots y_n & \left(\begin{array}{c} f(x_1,\dots,x_n) \doteq f(y_1,\dots,y_n) \\ \to x_1 \doteq y_1 \wedge \cdots \wedge x_n \doteq y_n \end{array} \right) & f \in \Sigma \\
(F_2) & \forall x_1 \cdots x_m y_1 \cdots y_n & f(x_1,\dots,x_m) \not\doteq g(y_1,\dots,y_n) & f, g \in \Sigma, f \not\equiv g \\
(F_3) & \qquad \forall x & (x \not\doteq t[x]) &
\end{array}
$$

where $t[x]$ denotes a term having x as a proper subterm

They are well-known as freeness axioms (or Clark equality axioms).

Axiom schemata (F_1) and (F_2) tell us, in particular, that

1. different constants denote different objects;
2. different functors are different data constructors;
3. constructed objects are equal (if and) only if they are constructed from equal components;
4. constructed objects are distinct from any constant.

They also impose that for any pair of *ground terms* x and t, if x is a subterm of t, then $x \neq t$. Axiom schema (F_3) enforces the last property by saying that data constructors always generate new objects. It is easy to see that if Σ contains at least one constant symbol, then any model of $\doteq_1 \doteq_2 F_1 F_2 F_3$ contains an isomorphic copy of the Herbrand Universe.

Axiom schema (F_3) states that there exists no term which is also subterm of itself. This is a form of well-foundedness. Since we deal also with membership, we must also require that a term t having x as a (proper or not) subterm cannot belong to x (hence, as a particular case, there cannot be cycles of memberships of the form $x_0 \in x_1 \in \ldots \in x_n \in x_0$). This can be expressed by axiom $(F_{(\notin)})$:

$(F_{(\notin)})$ $\quad \forall x \quad (t[x] \notin x)$
where $t[x]$ denotes a term having x as a proper subterm

In the following lemma it will be shown that in a sensible class of models of the theories we deal with, axiom $(F_{(\notin)})$ is superfluous.

Lemma 3.36 *In any model of KW whose domain is a subset of the Herbrand universe or of the complete Herbrand universe, then* (F_3) *implies* $(F_{(\notin)})$.

Proof. Let $M = \langle D, I \rangle$ be a model of $KW(DCA)$. Assume $t^I \in^I x^I$ and x be a subterm of t. *(DCA)* ensures that any object in D is the image of an ur-element or of a list. Axiom (K) and the fact that $t^I \in^I x^I$ ensures that x^I is the image of a list, hence it has the form $[\cdots, t, \cdots]^I$. Since $x \preceq t$ and $t \prec x$ then $x \prec x$: this contradicts (F_3).

3.36 □

The converse is not true: as an example, the rational tree that gives a solution to $x \doteq f(x)$ contradicts (F_3); however, axiom (K) ensures that $f(x) \notin x$.

Since we are interested exactly in models based on the Herbrand universe (standard or complete), we are allowed to forget the weaker non-membership freeness axiom.

If axiom schema (F_3) is removed, then a constraint of the form $x \doteq f(x)$ can be satisfied in the so-called complete Herbrand Universe (cf. Def. 2.4).

Axiomatization of the complete Herbrand universe

As said in § 2.1, the complete Herbrand universe is also made up of data structures whose algorithmic handling is not possible. We are interested in infinite elements which can be finitely represented, namely the rational terms (cf. Def. 2.5). Given a term t, one can 'fold' it by fusing two nodes μ, ν of $dom(t)$ into a single node whenever the subterms rooted at μ, ν are equivalent to each other. This will lead to a graph $\mathcal{G}$ retaining information of all essential features of t (the *picture* of t). If there are no infinite paths in $\mathcal{G}$, this indicates that the original t was already finite: this is the case of an ordinary term. When $\mathcal{G}$ is finite, t (which might be infinite) is said to be a *rational* term (cf. also Def 2.5).

For instance, provided that Σ contains the one-place functor symbol f, the term t_f defined as

- $dom(t_f) = \{[\underbrace{0, \dots, 0}_{n}] : n \geq 0\}$;
- $t_f(\nu) = f$, for all ν in $dom(t_f)$.

coincides with everyone of its subterms. Notice that it is a solution to the above constraint $x \doteq f(x)$. If Σ consists of the constant symbols $0, \dots, 9$, and of the binary symbol *cons*, the infinite term t_π defined as follows:

- $dom(t_\pi) = \{[\underbrace{1, \dots, 1}_{n}, 0], [\underbrace{1, \dots, 1}_{n}] : n \geq 0\}$;

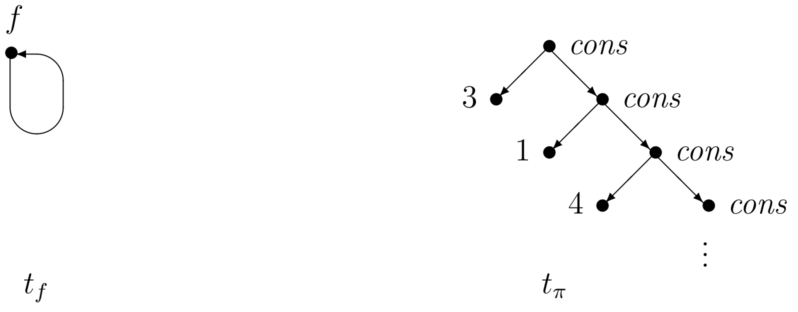


Figure 3.5: The picture of a rational and of an irrational term

- $t_\pi([\,]) = t_\pi([1, \dots, 1]) = cons$;
- $t_\pi([\underbrace{1, \dots, 1}_{i}, 0]) =$ the ith digit in the decimal expansion of π,

cannot be finitely represented. The two examples are depicted in Fig. 3.5.

Restricting the attention to rational terms suitable represented, then, even infinite terms can be algorithmically constructed and manipulated.

For any Herbrand system $\mathcal{E}$ of the form

$$\mathcal{E} \quad = \quad x_1 \doteq t_1 \wedge \dots \wedge x_n \doteq t_n \ ,$$

with x_i's distinct variables, when any variable

$$y_1, \dots, y_k \in FV(t_1, \dots, t_n) \setminus \{x_1, \dots, x_n\}$$

(we will refer to such variables as *eliminable* variables) is assigned to a term $t_1, \dots, t_k$ in the complete Herbrand universe, then a solution in the complete Herbrand Universe for the variables $x_1, \dots, x_n$ can be found in the following way:[7]

[7]Observe that the assumption '*x_i's distinct variables*' is necessary for the existence of the solution. To provide a counter-example, consider the system: $X_1 \doteq [\,] \wedge X_1 \doteq [Y_1, | \, Y_2]$. Freeness axiom ($F_2$) (or Lemma 3.1) ensures that it is unsatisfiable.

- Without loss of generality, we can assume that no equations of the form $X_i \doteq X_j$ $(i, j \in \{1, \dots, n\})$ are in $\mathcal{E}$. If this was not the case, we would apply the substitutions $[X_i/X_j]$ to $\mathcal{E}$, when $i \neq j$ and remove the identities $X_j \doteq X_j$ obtained. Correctness of such procedure is ensured by equality axioms $(\doteq_1)$ and $(\doteq_2)$.
- Moreover, still without loss of generality, we can assume that the system $\mathcal{E}$ is *flat*, namely every equation is of the form $X_i \doteq Y_j$ or $X_i \doteq f(V_1, \dots, V_{ar(f)})$, $V_1, \dots, V_{ar(f)}$ variables.
- We will build a (possibly infinite) graph G whose nodes are labeled by elements of Σ using a function T. After this definition, the rational term obtained by infinite unfolding of G will represent a solution (a tree-solution) for all $X_1, \dots, X_n$ satisfying $\mathcal{E}$.
  - $\nu_1, \dots, \nu_n$ are the nodes related to the variables $X_1, \dots, X_n$.
  - for any equation $X_i \doteq f(V_1, \dots, V_{ar(f)})$, let $T(\nu_i) = f$, and add the edges
  $$\langle \nu_i, \mu_1 \rangle, \dots, \langle \nu_i, \mu_{ar(f)} \rangle$$
  to G, where μ_j is
    * ν_k if V_j is X_k,
    * the root of the term t_k assigned to Y_k if V_j is the eliminable variable Y_k.

The solution for a variable X_i is the subterm of the term obtained by infinite unfolding of G rooted at ν_i. Intuitively, the solution obtained in this way is also the *unique* solution in the complete Herbrand universe. Maher, in [74], introduces explicitly axiom schema (F_4):

$$(F_4) \qquad \forall y_1 \cdots y_m \exists! x_1 \cdots x_n \, (x_1 \doteq t_1 \wedge \cdots \wedge x_n \doteq t_n)$$
$$x_i\text{'s are pairwise distinct variables,}$$
$$FV(t_i) \subseteq \{x_1, \dots, x_n, y_1, \dots, y_n\}$$

This allows him to prove the *completeness* of the theory consisting of axioms (F_1), (F_2), (F_4), and (if the signature is finite is needed, otherwise it can be removed) (DCA), when $\Pi = \{\doteq\}$ (cf. § 3.1.1). For instance, the system $X_1 \doteq f(X_2) \wedge X_2 \doteq f(X_1)$ admits the unique (tree) solution $X_1 = f(f(f(\cdots))), X_2 = f(f(f(\cdots)))$, which implies $X_1 = X_2$.

Remark 3.37 *Notice that if the axiomatization were less restrictive, it would be possible to find solutions in which X_1 is distinct from X_2. For instance, assume that the domain of the structure in which we can find solutions is the set of natural numbers and that it is consistent with the theory that f is defined as follows:*

$$f(x) = \begin{cases} x-1 & \text{if } x \text{ is odd} \\ x+1 & \text{if } x \text{ is even} \end{cases}$$

Then, for instance, $X_1 \equiv 0(= f(1))$ and $X_2 = 1(= f(0))$ is one of the (infinite) solutions for X_1 and X_2 in which $X_1 \neq X_2$.

All this reasoning has a (famous) counter part in pure set theory. Anti-foundation axiom **AFA** was introduced by Aczel in [3] to extend extensionality axioms when the sets are non-well-founded. Intuitively, a set is non-well-founded when it has an infinite descending membership sequence; i.e. an infinite sequence of sets, consisting of an element of the set, an element of that element, an element of that element of that element and so on ad infinitum (for an analysis of the concept of well-foundedness, see § A.2). Aczel's sets are nodes of graphs whose edges represents the membership relation. The infinite unfolding of such graphs produces a term of a signature having a set constructor $\underbrace{\{\cdot, \ldots, \cdot\}}_{n}$ of arity n for each n. Given a term of this form, it is easy to rewrite it in a term written in the signature $\{\emptyset, \{\cdot \mid \cdot\}\}$. It is not surprising, therefore, to be able to adapt the anti-foundation axiom **AFA**: *Every graph has a unique decoration* to our notation, in order to establish when a system of equations between sets has a solution and, if this is the case, how many solutions there are.

Given a system of equations

$$\mathcal{E} \quad = \quad X_1 \doteq t_1 \wedge \ldots \wedge X_n \doteq t_n$$

where $X_1, \ldots, X_n$ are pairwise distinct variables and $FV(t_1, \ldots, t_n) = \{X_1, \ldots, X_n, Y_1, \ldots, Y_n\}$, then, once an assignment to a set for the variables $Y_1, \ldots, Y_n$ is fixed,

(AFA 1) every system of the form above admits a solution; moreover

(AFA 2) such a solution is unique.

In the following four subsections we will properly adapt the above axioms so as to fulfill the intended meaning of the list, bag, compact list, and set constructor symbols.

Zippers and infinite objects

Before enter into the details of the various hybrid aggregate theories that will be axiomatically presented, we want to point out the different behavior of the aggregate data structures analyzed with respect to particular sets of equations. We will see that such systems contains enough information to force the infiniteness of the solution to all the cases but the sets one. Moreover, we will point out that infinite (rational) solution and non-well-founded solutions are independent concepts.

Consider the following unification problem(s)

$$\begin{array}{ll} (\ell_1) & X \doteq [[\,]\,|\,X] \\ (m_1) & X \doteq \{\!\!\{\, \{\!\!\{\ \}\!\!\}\,|\,X\,\}\!\!\} \\ (c_1) & X \doteq [\![\, [\![\ ]\!]\,|\,X\,]\!] \\ (s_1) & X \doteq \{\emptyset\,|\,X\} \end{array}$$

The (unique) solution to the list case (ℓ_1) is the infinite list $X = [[\,], [\,], [\,], \ldots]$. The corresponding bindings

$$X = \{\!\!\{\, \{\!\!\{\ \}\!\!\}, \{\!\!\{\ \}\!\!\}, \{\!\!\{\ \}\!\!\}, \ldots \}\!\!\},\ X = [\![\, [\![\ ]\!], [\![\ ]\!], [\![\ ]\!], \ldots]\!],\ X = \{\emptyset, \emptyset, \emptyset, \ldots\}$$

are solutions to the problems (m_1), (c_1), and (s_1), respectively. Nevertheless, they are not the unique solution (neither the more general). Notice that, although involving infinite aggregates, such solutions do not break well-foundedness of membership.

Any bag containing an infinite number of $\{\!\!\{\ \}\!\!\}$ is a solution to (m_1); furthermore, any compact list *beginning* with the element $[\![\ ]\!]$ is a solution to (c_1), and any set containing the element $\emptyset$ is a solution to problem (s_1). Observe that lists and bags admits only infinite solutions to this problem.

Consider now the problem:

$$\begin{array}{llcl} (\ell_2) & X_0 \doteq [[\,]\,|\,X_1] & \wedge & X_1 \doteq [X_0\,|\,X_0] \\ (m_2) & X_0 \doteq \{\!\!\{\, \{\!\!\{\ \}\!\!\}\,|\,X_1\,\}\!\!\} & \wedge & X_1 \doteq \{\!\!\{\, X_0\,|\,X_0\,\}\!\!\} \\ (c_2) & X_0 \doteq [\![\, [\![\ ]\!]\,|\,X_1\,]\!] & \wedge & X_1 \doteq [\![\, X_0\,|\,X_0\,]\!] \\ (s_2) & X_0 \doteq \{\emptyset\,|\,X_1\} & \wedge & X_1 \doteq \{X_0\,|\,X_0\} \end{array}$$

Since X_0 and X_1 are both forced to contain elements, they are distinct from the empty entities in all the solution to the above problems.

The (unique) solution to (ℓ_2) is the—implicitly described—binding

$$X_0 = [[\,], X_0, [\,], X_0, [\,], X_0, \ldots], X_1 = [X_0, [\,], X_0, [\,], X_0, [\,], \ldots],$$

for which, in particular, we can infer that $X_0 \neq X_1$.

(m_2) forces $X_0 = X_1$ in any solution. Moreover, any solution must associate to X_0 a bag with an infinite number of occurrences of $\{\!\!\{\ \}\!\!\}$ and X_0 itself.

(c_2) admits the (unique) solution:

$$\begin{array}{rcl} X_0 & = & [\![[\![\ ]\!], X_0, [\![\ ]\!], X_0, [\![\ ]\!], X_0, \ldots]\!], \\ X_1 & = & [\![X_0, [\![\ ]\!], X_0, [\![\ ]\!], X_0, [\![\ ]\!], \ldots]\!], \end{array}$$

which forces X_0 to be distinct from X_1.

The binding $X_0 = \{\emptyset, X_0 \,|\, N\}, X_1 = X_0$, where N is a newly generated variable, is the most general solution to the problem (s_2). Any term (finite or infinite) associated to such variable fulfills the requirements of the problem.

Observe that this second problem forces all the solutions to be infinite, save for the set case. Moreover, in [90, 91], it is shown that the simplest way to force a formula to be satisfied by non-finite sets require a logical complexity greater than a simple equation system.

However, notice that well-foundedness of membership is violated in all the solutions to this problem, since X_0 is forced to belong to itself.

Subsystems generating situations of the form above are called *zippers*:

Definition 3.38 *A conjunction of equalities of the form*

$$X_0 \doteq [Y_0 \,|\, X_1] \wedge \ldots \wedge X_{m-1} \doteq [Y_{m-1} \,|\, X_m] \wedge X_m \doteq [Y_m \,|\, X_0]$$

(similarly for $\{\!\!\{ \cdot \,|\, \cdot \}\!\!\}$, $[\![\cdot \,|\, \cdot]\!]$, and $\{ \cdot \,|\, \cdot \}$) is called a Zipper. *For $m = 0$ this reduces to the single equation $X_0 \doteq [Y_0 \,|\, X_0]$.*

Zippers, which force infiniteness for all the aggregate data structures save the set one, will play crucial roles both in the axiomatizations described in the following sections and in the unification algorithms that will be presented in Chapter 4.

3.2.1 Lists

We need to integrate the so-called *freeness axioms*, developed for uninterpreted terms, with the interpreted functional symbol $[\cdot\,|\,\cdot]$.

As already stated, axiom (F_2) is a theorem of KW when one of the two symbols f, g is $[\cdot\,|\,\cdot]$.

Since the number of occurrences and the ordering of elements in a list are both important, axiom (F_1) must be extended for the functor $[\cdot\,|\,\cdot]$. In the remaining sections of this chapter, we will discover that under axioms (E_1) and/or (E_2) this is no longer true. Moreover, under absorption axiom (E_2), axiom schema (F_3) cannot be accepted.

The axioms identifying the *well-founded hybrid theory of lists* (`WF_lists`), where $\Pi = \{\doteq, \in\}$ and $\Sigma = \{[\,], [\cdot\,|\,\cdot], \dots\}$, are the equality axioms $(\doteq_1)$, $(\doteq_2)$, plus

(K)	$\forall x\, y_1 \cdots y_n$	$(x \notin f(y_1, \dots, y_n))$	$f \in \Sigma \setminus \{[\cdot\,|\,\cdot]\}$
(W)	$\forall xyz$	$(x \in [y\,|\,z] \leftrightarrow x \doteq y \vee x \in z)$	
(F_1)	$\forall x_1 \cdots x_n y_1 \cdots y_n$	$\left(\begin{array}{c} f(x_1, \dots, x_n) \doteq f(y_1, \dots, y_n) \\ \to x_1 \doteq y_1 \wedge \cdots \wedge x_n \doteq y_n) \end{array}\right)$	$f \in \Sigma$
(F_2)	$\forall x_1 \cdots x_m y_1 \cdots y_n$	$f(x_1, \dots, x_m) \not\doteq g(y_1, \dots, y_n)$	$f, g \in \Sigma, f \not\equiv g$
(F_3)	$\forall x$	$x \not\doteq t[x]$	

To obtain a non-well founded theory, axiom (F_3) must be suitably replaced by another one. Assume $x = t[x]$: it has the unique solution $t[t[t[t[\cdots]]]]$ in the complete Herbrand Universe. Hence axiom (F_3) is replaced by axiom

$$
\begin{array}{ll}
(F_4) & \forall y_1 \cdots y_m \exists! x_1 \cdots x_n\, (x_1 \doteq t_1 \wedge \cdots \wedge x_n \doteq t_n) \\
 & x_i\text{'s are pairwise distinct variables,} \\
 & FV(t_i) \subseteq \{x_1, \dots, x_n, y_1, \dots, y_n\}
\end{array}
$$

This theory allows to deal with finite and infinite well-founded lists, as well as for finite and infinite non-well-founded-lists. It is in fact immediate to see that it provides (tree) solutions to the equations:

- $x \doteq [[\,]\,|\,x]$ (which has the infinite and well-founded tree solution $x = [[\,], [\,], [\,], \dots])$,
- $x \doteq [x]$ (which has the finite and non-well-founded tree solution $x = [[[[\cdots]]]]$), and

- $x \doteq [x \mid x]$ (which has the infinite and non-well-founded tree solution implicitly described by $x = [x, x, x, \ldots]$).

We will refer as `NWF_lists` to this theory.[8]

3.2.2 Multi sets

The signature of a hybrid theory of bags must comprise the binary functional symbol $\{\!\!\{ \cdot \mid \cdot \}\!\!\}$, whose behavior is regulated by the permutativity axiom (E_1) (see § 3.1.2). This means, in particular, that

$$\{\!\!\{ a \mid \{\!\!\{ b \}\!\!\} \}\!\!\} \ (i.e.\ \{\!\!\{ a, b \}\!\!\}) \quad \doteq \quad \{\!\!\{ b \mid \{\!\!\{ a \}\!\!\} \}\!\!\} \ (i.e.\ \{\!\!\{ b, a \}\!\!\})$$

even if a is distinct from b. Hence axiom schema (F_1) does not hold when f is instantiated to $\{\!\!\{ \cdot \mid \cdot \}\!\!\}$. Therefore, we introduce a weaker axiom (F_1'), defined as

$$(F_1') \qquad \forall x_1 \cdots x_n y_1 \cdots y_n \begin{pmatrix} f(x_1, \ldots, x_n) \doteq f(y_1, \ldots, y_n) \\ \rightarrow x_1 \doteq y_1 \wedge \cdots \wedge x_n \doteq y_n) \end{pmatrix}$$

only when $f \in \Sigma \setminus \{\{\!\!\{ \cdot \mid \cdot \}\!\!\}\}$. On the other hand, we need to establish a criterion for stating when (and only when) two (hybrid) bags are to be considered equal; following the ideas presented in § 3.1.2 we can introduce axiom (E_k^m):

$$(E_k^m) \qquad \forall y_1 y_2 v_1 v_2 \begin{pmatrix} \{\!\!\{ y_1 \mid v_1 \}\!\!\} \doteq \{\!\!\{ y_2 \mid v_2 \}\!\!\} \ \leftrightarrow \\ (y_1 \doteq y_2 \wedge v_1 \doteq v_2) \vee \\ \exists k\, (v_1 \doteq \{\!\!\{ y_2 \mid k \}\!\!\} \wedge v_2 \doteq \{\!\!\{ y_1 \mid k \}\!\!\}) \end{pmatrix}$$

or we can enrich Π with an infinite number of predicate symbols $\in^n$, one for every natural number n, and use the multiset equality principle, modified here in order to capture the concept of kernel:

$$(E^m) \qquad \forall v_1 v_2 \text{ for all } n\ \forall x \begin{pmatrix} (x \in^n v_1 \leftrightarrow x \in^n v_2) \\ \wedge ker(v_1) \doteq ker(v_2) \end{pmatrix} \rightarrow v_1 \doteq v_2$$

Since the two extensions can be considered equivalent for finite bags (it is easy to extend the proof of Theorem 3.19 to the hybrid case), we prefer to choose the first approach, which allows Π left unchanged

[8]Observe that `WF_lists` and `NWF_lists` are Maher's axioms from [74] plus the axioms (K) and (W) to govern membership.

(and finite) and it follows the parametric guideline of the thesis: axiom (E_k^m) extends axiom (F_1) used for lists with a disjunct on the right of the double impication.

We show here that the '$\leftarrow$' direction of axiom (E_k^m) implies axiom (E_1) that can therefore be removed.

Lemma 3.39 $(E_k^m) \Rightarrow (E_1)$

Proof. Consider the two bag-terms $\{\!\!\{\, x, y \mid z \,\}\!\!\}$ and $\{\!\!\{\, y, x \mid z \,\}\!\!\}$. If $x \doteq y$ they are equal for $(\doteq_1)$. Assume $x \not\doteq y$; by (E_k^m) $\{\!\!\{\, x, y \mid z \,\}\!\!\} \doteq \{\!\!\{\, y, x \mid z \,\}\!\!\}$ if and only if $\exists k\, (\{\!\!\{\, y \mid z \,\}\!\!\} \doteq \{\!\!\{\, y \mid k \,\}\!\!\} \wedge \{\!\!\{\, x \mid z \,\}\!\!\} \doteq \{\!\!\{\, x \mid k \,\}\!\!\})$. Choosing k as 'z' the result follows.

3.39 □

The axioms identifying the *well-founded hybrid theory of multisets (*`WF_bags`*)*, where $\Pi = \{\doteq, \in\}$ and $\Sigma = \{\{\!\!\{\ \}\!\!\}, \{\!\!\{\, \cdot \mid \cdot \,\}\!\!\}, \ldots\}$, are therefore $(\doteq_1)$, $(\doteq_2)$, plus

(K)	$\forall x\, y_1 \cdots y_n$	$(x \notin f(y_1, \ldots, y_n))\quad f \in \Sigma \setminus \{\{\!\!\{\, \cdot \mid \cdot \,\}\!\!\}\}$
(W)	$\forall xyz$	$(x \in \{\!\!\{\, y \mid z \,\}\!\!\} \leftrightarrow x \doteq y \vee x \in z)$
(E_k^m)	$\forall y_1 y_2 v_1 v_2$	$\left(\begin{array}{c} \{\!\!\{\, y_1 \mid v_1 \,\}\!\!\} \doteq \{\!\!\{\, y_2 \mid v_2 \,\}\!\!\} \ \leftrightarrow \\ (y_1 \doteq y_2 \wedge v_1 \doteq v_2) \vee \\ \exists k\, (v_1 \doteq \{\!\!\{\, y_2 \mid k \,\}\!\!\} \wedge v_2 \doteq \{\!\!\{\, y_1 \mid k \,\}\!\!\}) \end{array}\right)$
(F_1')	$\forall x_1 \cdots x_n y_1 \cdots y_n$	$\left(\begin{array}{l} f(x_1, \ldots, x_n) \doteq f(y_1, \ldots, y_n) \\ \rightarrow x_1 \doteq y_1 \wedge \cdots \wedge x_n \doteq y_n) \end{array}\right)$ $f \in \Sigma \setminus \{\{\!\!\{\, \cdot \mid \cdot \,\}\!\!\}\}$
(F_2)	$\forall x_1 \cdots x_m y_1 \cdots y_n$	$f(x_1, \ldots, x_m) \not\doteq g(y_1, \ldots, y_n)$ $f, g \in \Sigma, f \not\equiv g$
(F_3)	$\forall x$	$x \not\doteq t[x]$

For the non-well founded case, similarly to what was shown for hybrid lists, we will replace axiom schema (F_3) by a suitable counterpart of the anti-foundation axiom **(AFA)**. Nevertheless, a *zipper*—see Def. 3.38—is a template denoting infinite solutions (for instance, $X = \{\!\!\{\, a \mid X \,\}\!\!\}$: any bag containing the constant term 'a' an infinite number of times is a solution for X). This means that the uniqueness requirement of the axiom must be dropped:

(F_4^m) $\quad \forall y_1 \cdots y_m \exists x_1 \cdots x_n\, (x_1 \doteq t_1 \wedge \cdots \wedge x_n \doteq t_n)$
x_i's are pairwise distinct variables,
$FV(t_i) \subseteq \{x_1, \ldots, x_n, y_1, \ldots, y_n\}$

One might replace axiom (E^m_k) (proved to be equivalent to axiom (E^m)) with an axiom capturing the concept of *infiniteness*:

$$(E^m_{ord}) \qquad \forall v_1 v_2 \text{ for all } \alpha \, \forall x \left(\begin{array}{c} (x \in^\alpha v_1 \leftrightarrow x \in^\alpha v_2) \\ \wedge ker(v_1) \doteq ker(v_2) \end{array} \right) \rightarrow v_1 \doteq v_2$$

where α ranges over ordinal numbers (see, e.g., [64]). However, this requires the introduction of ordinal numbers in the theory (or in the meta-theory) and to extend signature and theory in order to deal with *kernel* entities.

Since we are interested in infinite but finitely representable bags (namely, via systems of equations), the former requirement can be simplified as follows:

$$(E^m_\omega) \quad \forall v_1 v_2 \quad \text{for all } n \, \forall x \left(\begin{array}{c} (x \in^n v_1 \leftrightarrow x \in^n v_2) \wedge \\ (x \in^\omega v_1 \leftrightarrow x \in^\omega v_2) \wedge \\ ker(v_1) \doteq ker(v_2) \end{array} \right) \rightarrow v_1 \doteq v_2$$

where n ranges over natural numbers, the $\in^n$'s are defined in § 3.1.2, and $x \in^\omega y$ is a shorthand for the formula $y \doteq \{\!\!\{ x \,|\, y \}\!\!\}$.

The functional symbol *ker* can be seen as a meta-symbol when the bags are finite. Otherwise new axioms concering with it must be introduced. However, as it will show in Theorem 4.49, axiom (E^m_k) (which is clearly implied by (E^m_ω)) contains sufficient information to simplify Herbrand system in an equivalent form. We therefore leave it in the theory.

Such theory will be denoted by the abbreviation `NWF_bags`.

3.2.3 Compact lists

Similarly to bags, also for compact lists axiom schema (F_1) does not correctly model the behavior of functional symbol $[\![\cdot \,|\, \cdot]\!]$ (which is regulated by the absorption axiom (E_2)—see § 3.1.3), as it ensues from the example:

$$[\![a \,|\, [\![a]\!]]\!] \, (i.e. \, [\![a, a]\!]) \quad \doteq \quad [\![a \,|\, [\![\;]\!]]\!] \, (i.e. \, [\![a]\!])$$

Notice that $[\![a]\!]$ is distinct from $[\![\;]\!]$.

Moreover, also freeness axiom (F_3) must be modified accordingly to the interpretation of $[\![\cdot \,|\, \cdot]\!]$. In the just described example, for instance,

$[\![a]\!]$ is a proper subterm of $[\![a \,|\, [\![a]\!]]\!]$; nevertheless, they are equal in the theory (and also the desired well-foundedness holds—$x \doteq [\![a \,|\, x]\!]$ admits a finite tree solution).

We first recall the equality principle for compact lists (with kernels) presented in § 3.1.3:

$$(E_k^c) \qquad \forall y_1 y_2 v_1 v_2 \left(\begin{array}{l} [\![y_1 \,|\, v_1]\!] \doteq [\![y_2 \,|\, v_2]\!] \ \leftrightarrow \\ \qquad (y_1 \doteq y_2 \wedge v_1 \doteq v_2) \vee \\ \qquad (y_1 \doteq y_2 \wedge v_1 \doteq [\![y_2 \,|\, v_2]\!]) \vee \\ \qquad (y_1 \doteq y_2 \wedge [\![y_1 \,|\, v_1]\!] \doteq v_2) \end{array} \right)$$

that, in particular, makes axiom (E_2) superfluous.

Let c be the equational theory consisting of axiom (E_2); axiom (F_3^c) must keep into account the result stated in Lemma 3.21:

(F_3^c) $\quad x \neq t[x]$
unless t is of the form $[\![t_0, \dots, t_n \,|\, x]\!]$,
and $t_0 \doteq \dots \doteq t_n$ and x does not occur in $t_0 \dots t_n$

The axioms below, together with equality axioms $(\doteq_1)$ and $(\doteq_2)$ identify the *well-founded hybrid theory of compact lists* WF_clists, where $\Pi = \{\doteq, \in\}$ and $\Sigma = \{[\![\ ]\!], [\![\cdot \,|\, \cdot]\!], \dots\}$.

(K)	$\forall x\, y_1 \cdots y_n$	$(x \notin f(y_1, \dots, y_n))\quad f \in \Sigma \setminus \{[\![\cdot \,|\, \cdot]\!]\}$
(W)	$\forall xyz$	$(x \in [\![y \,|\, z]\!] \leftrightarrow x \doteq y \vee x \in z)$
(E_k^c)	$\forall y_1 y_2 v_1 v_2$	$\left(\begin{array}{l} [\![y_1 \,|\, v_1]\!] \doteq [\![y_2 \,|\, v_2]\!] \ \leftrightarrow \\ \qquad (y_1 \doteq y_2 \wedge v_1 \doteq v_2) \vee \\ \qquad (y_1 \doteq y_2 \wedge v_1 \doteq [\![y_2 \,|\, v_2]\!]) \vee \\ \qquad (y_1 \doteq y_2 \wedge [\![y_1 \,|\, v_1]\!] \doteq v_2) \end{array} \right)$
(F_1')	$\forall x_1 \cdots x_n y_1 \cdots y_n$	$\left(\begin{array}{l} f(x_1, \dots, x_n) \doteq f(y_1, \dots, y_n) \\ \to x_1 \doteq y_1 \wedge \cdots \wedge x_n \doteq y_n) \end{array} \right)$
		$f \in \Sigma \setminus \{[\![\cdot \,|\, \cdot]\!]\}$
(F_2)	$\forall x_1 \cdots x_m y_1 \cdots y_n$	$f(x_1, \dots, x_m) \neq g(y_1, \dots, y_n)\ \ f, g \in \Sigma, f \not\equiv g$
(F_3^c)	$\forall x$	$x \neq t[x]$
		unless t is of the form $[\![t_0, \dots, t_n \,|\, x]\!]$,
		$t_0 \doteq \dots \doteq t_n$, and
		x does not occur in $t_0 \dots t_n$

For the non-well-founded case, after removing axiom (F_3^c), consider the satisfiability problem for the constraint $x \doteq [\![\, y_1, y_2 \,|\, x \,]\!]$ in the complete Herbrand universe. If y_1 is distinct from y_2, then the unique solution is the (rational) infinite term $[\![\, y_1, y_2, y_1, y_2, \cdots]\!]$. If y_1 is equal to y_2 it admits the solution $x = [\![\, y_1 \,|\, N \,]\!]$, for any Σ-term instantiation for the variable N. Clearly, such a solution is not unique.

The same problem occurs when we consider the satisfiability problem of any ZIPPER (cf. Def. 3.38). This means that also for compact lists the uniqueness requirement of the axiom must be dropped:

$$(F_4^c) \qquad \begin{array}{l} \forall y_1 \cdots y_m \exists x_1 \cdots x_n \, (x_1 \doteq t_1 \wedge \cdots \wedge x_n \doteq t_n) \\ x_i\text{'s are pairwise distinct variables,} \\ FV(t_i) \subseteq \{x_1, \ldots, x_n, y_1, \ldots, y_n\} \end{array}$$

The theory of (even infinite) non-well-founded compact-lists will be denoted by `NWF_clists`.

3.2.4 Sets

As shown in § 3.1.4, the insertion of both axioms (E_1) and (E_2) generates a minimal theory of sets. The introduction of (free) functors into the signature generates the hybrid theory of sets, a framework suited for high level declarative programming, as it will be shown in Chapter 7. Similarly to what done in the previous three subsections, we choose one extensionality criterion for testing the equality of two sets:

$$(E_k^s) \qquad \forall y_1 y_2 v_1 v_2 \left(\begin{array}{c} \{y_1 \,|\, v_1\} \doteq \{y_2 \,|\, v_2\} \;\; \leftrightarrow \\ (y_1 \doteq y_2 \wedge v_1 \doteq v_2) \vee \\ (y_1 \doteq y_2 \wedge v_1 \doteq \{y_2 \,|\, v_2\}) \vee \\ (y_1 \doteq y_2 \wedge \{y_1 \,|\, v_1\} \doteq v_2) \vee \\ \exists k \, (v_1 \doteq \{y_2 \,|\, k\} \wedge v_2 \doteq \{y_1 \,|\, k\}) \end{array} \right)$$

which is shown to imply both (E_1) and (E_2) that can, henceforth, be ignored. Moreover, a simple extension of the classical extensionality axiom with the meta-concept of kernel is the following:

$$(E^s) \qquad \forall v_1 v_2 \;\; \forall x \left(\begin{array}{c} (x \in v_1 \leftrightarrow x \in v_2) \wedge \\ ker(v_1) \doteq ker(v_2) \end{array} \right) \rightarrow v_1 \doteq v_2$$

(it will be useful in the next chapter, in handling negative constraints). It is easy to modify the proofs of Theorems 3.27 and 3.33 to show that it is equivalent to (E^s_k) in all models of finite sets (that are sufficient for our purposes), hence we will use such axiom but we only need to introduce the parametric axiom (E^s_k) in the theory.

Axiom schema (F^s_3) for sets simplifies axiom schema (F^c_3) introduced for compact lists:

(F^s_3) $\quad X \not\doteq t[x]$
unless $t[x]$ is of the form $\{t_0, \dots, t_n \,|\, x\}$,
and x does not occur in $t_0, \dots, t_n$.

The axioms identifying the *well-founded hybrid theory of sets* (`WF_sets`), where $\Pi = \{\doteq, \in\}$ and $\Sigma = \{[\,], \{\,\cdot\,|\,\cdot\,\}, \dots\}$, are $(\doteq_1)$, $(\doteq_2)$, plus

(K)	$\forall x\, y_1 \cdots y_n$	$(x \notin f(y_1, \dots, y_n)) \quad f \in \Sigma \setminus \{\{\cdot\,|\,\cdot\}\}$
(W)	$\forall xyz$	$(x \in \{y \,|\, z\} \leftrightarrow x \doteq y \vee x \in z)$
(E^s_k)	$\forall y_1 y_2 v_1 v_2$	$\left(\begin{array}{l} \{y_1 \,|\, v_1\} \doteq \{y_2 \,|\, v_2\} \;\leftrightarrow \\ \qquad (y_1 \doteq y_2 \wedge v_1 \doteq v_2) \vee \\ \qquad (y_1 \doteq y_2 \wedge v_1 \doteq \{y_2 \,|\, v_2\}) \vee \\ \qquad (y_1 \doteq y_2 \wedge \{y_1 \,|\, v_1\} \doteq v_2) \vee \\ \quad \exists k\, (v_1 \doteq \{y_2 \,|\, k\} \wedge v_2 \doteq \{y_1 \,|\, k\}) \end{array}\right)$
(F'_1)	$\forall x_1 \cdots x_n y_1 \cdots y_n$	$(f(x_1, \dots, x_n) \doteq f(y_1, \dots, y_n) \rightarrow$ $x_1 \doteq y_1 \wedge \cdots \wedge x_n \doteq y_n) \quad f \in \Sigma \setminus \{\{\cdot\,|\,\cdot\}\}$
(F_2)	$\forall x_1 \cdots x_m y_1 \cdots y_n$	$f(x_1, \dots, x_m) \not\doteq g(y_1, \dots, y_n)$ $f, g \in \Sigma \setminus \{\{\cdot\,|\,\cdot\}\}, f \not\equiv g$
(F^s_3)	$\forall x$	$x \not\doteq t[x]$ unless $t[x]$ is of the form $\{t_0, \dots, t_n \,|\, x\}$, and x does not occur in $t_0, \dots, t_n$.

Removing axiom (F^s_3) allows a non-well-founded interpretation of the theory; an interesting example of purely set-theoretic irrational term is the one whose picture is the graph of Fig. 3.3.

As shown in § 3.2.3, axiom (E_2) causes the uniqueness (once parameters has been instantiated) of the solution to a system of equations in solved form is no longer true. However, using the pre-processing process:

$$\left.\begin{array}{lcl} v_0 & \doteq & \{t^0_0,\dots,t^0_{k_0}\,|\,v_1\},\\ v_1 & \doteq & \{t^1_0,\dots,t^1_{k_1}\,|\,v_2\},\\ \dots & & \\ v_n & \doteq & \{t^n_0,\dots,t^n_{k_n}\,|\,v_0\} \end{array}\right\} \mapsto \left\{\begin{array}{lcl} v_0 & \doteq & \{t^0_0,\dots,t^0_{k_0},\\ & & t^1_0,\dots,t^1_{k_1},\\ & & \dots,\\ & & t^n_0,\dots,t^n_{k_n}\,|\,z\},\\ v_1 & \doteq & v_0,\\ \vdots & \vdots & \vdots\\ v_n & \doteq & v_0 \end{array}\right.$$

where $z_0,\dots,z_n$ are new variables, the uniqueness is again guaranteed. The adapted version of axiom schema (F_4), named (F_4^s), taking the place of standard **AFA** axiom (see [3]) is the following:

(F_4^s) $\forall y_1 \cdots y_m \exists!\, x_1 \cdots x_n\, (x_1 \doteq t_1 \wedge \cdots \wedge x_n \doteq t_n)$
x_i's are pairwise distinct variables,
there are no zippers in $x_1 \doteq t_1 \wedge \cdots \wedge x_n \doteq t_n$
$FV(t_i) \subseteq \{x_1,\dots,x_n,y_1,\dots,y_n\}$

The equality criterion between sets is equivalent to the classical extensionality axiom (enriched with a test of the equality of the kernels) when the two sets are finite (cf. § 3.1.4). It can be used for infinite sets by adding the removal axiom (L) to the theory. However, since we will be interested in sets defined by a (finite) system of equations, and since it is impossible to force infinite set solutions with a finite system of equations (cf. [90, 91]), the introduction of axiom (L) is not necessary.

The theory presented is named `NWF_sets`.

Fig. 3.6 summarizes the axiomatic theories presented in this section.

`WF_lists`	$(\dot{=})(K)(W)(F_1)(F_2)(F_3)$
`NWF_lists`	$(\dot{=})(K)(W)(F_1)(F_2)(F_4)$
`WF_bags`	$(\dot{=})(K)(W)(E_k^m)(F_1')(F_2)(F_3)$
`NWF_bags`	$(\dot{=})(K)(W)(E_k^m)(F_1')(F_2)(F_4^m)$
`WF_clists`	$(\dot{=})(K)(W)(E_k^c)(F_1')(F_2)(F_3^c)$
`NWF_clists`	$(\dot{=})(K)(W)(E_k^c)(F_1')(F_2)(F_4^c)$
`WF_sets`	$(\dot{=})(K)(W)(E_k^s)(F_1')(F_2)(F_3)$
`NWF_sets`	$(\dot{=})(K)(W)(E_k^s)(F_1')(F_2)(F_4^s)$

Figure 3.6: Hybrid theories

Chapter 4

Unification

In this chapter we will study the unification problems for the well-founded theories `WF_lists`, `WF_bags`, `WF_clists`, and `WF_sets` (§ 4.2), as well as for the non well-founded theories `NWF_lists`, `NWF_bags`, `NWF_clists`, and `NWF_sets` (§ 4.3). The parametric definition of the axiomatizations will allow a parametric development of such algorithms.

In the preliminary section 4.1, such unification problems are shown to be NP-hard, save the list one. Unification algorithms for non-well-founded theories of compact lists (§ 4.3.2), sets (§ 4.3.3), and bags (§ 4.3.4) can easily be modified to obtain an NP-algorithm for the corresponding well-founded problem.

The unification algorithm for hybrid and well-founded sets of § 4.2.4 was firstly presented in [37], as an extension of the one presented in [57] in a slight different context, to complete an inferential engine to the logic programming language with sets {log}. It was later implemented (cf. [40]) on a suitable extension of the Warren Abstract Machine, called {WAM}. The reduction to 3-SAT of the set unification problem, argument of § 4.1.3, was firstly presented in [38].

The hybrid hyperset unification algorithm follows from a stream of preliminary works which involved several people (cf. [7, 88]). The first organic presentation of that material is [35].

The minimality analysis, presented in § 4.4, is a summary of [11].

4.1 Complexity bounds

While unification for (hybrid) lists can be performed in linear time (see § 4.2.1), all other unification problems analyzed in this thesis are NP-complete.

To show NP-hardness we reduce the NP-complete problem 3-SAT to the unification problem for bags (§ 4.1.1), compact lists (§ 4.1.2), and sets (§ 4.1.3). First we recall 3-satisfiability (3-SAT—cf. [45]) problem:

Instance: A collection $C = \{\ell_1^1 \vee \ell_2^1 \vee \ell_3^1, \dots, \ell_1^m \vee \ell_2^m \vee \ell_3^m\}$ of clauses on a finite set $U = \{X_1, \dots, X_n\}$ of propositional letters, such that every ℓ_i^j $(i = 1, 2, 3; j = 1, \dots, m)$ is either of the form X_k or of the form $\neg X_k$, for one $k \in \{1, \dots, n\}$.

Question: Is there a truth assignment for U that satisfies all the clauses in C?

We will use variables $A_1, \dots, A_m, B_1, \dots, B_m, X_1, \dots, X_n, Y_1, \dots, Y_n$ $(|\,C\,| = m$ and $|\,V\,| = n)$. $\{\!\![\;]\!\!\}$ $([\![\;]\!]$, $\emptyset)$ will represent `false` and $\{\!\![\,\{\!\![\;]\!\!\}\,]\!\!\}$ $([\![\,[\![\;]\!]\,]\!]$, $\{\emptyset\})$ will represent `true`. The linear-time encoding of an instance $\langle C, U\rangle$ of 3-SAT into an instance Φ of the set unification problem is based on the function f defined as follows:

$$\begin{cases} f(X_i) & = & X_i \\ f(\neg X_i) & = & Y_i\,. \end{cases}$$

4.1.1 NP-hardness of multi-set unification

Any instance of 3-SAT can be mapped into the conjunction of bag unification problems

$$\begin{array}{rcll} \{\!\![\, X_i, Y_i \,]\!\!\} & \doteq & \{\!\![\, \texttt{false}, \texttt{true} \,]\!\!\} & i \doteq 1, \dots, n \\ \{\!\![\, f(\ell_1^j), f(\ell_2^j), f(\ell_3^j) \,]\!\!\} & \doteq & \{\!\![\, \texttt{true}, A_j, B_j \,]\!\!\} & j = 1, \dots, m \end{array}$$

or into the unique unification problem

$$\begin{array}{ll} \{\!\![& \{\!\![\, X_1, Y_1 \,]\!\!\}, \dots, \{\!\![\, X_n, Y_n \,]\!\!\}, \\ & \{\!\![\, f(\ell_1^1), f(\ell_2^1), f(\ell_3^1) \,]\!\!\}, \dots, \{\!\![\, f(\ell_1^m), f(\ell_2^m), f(\ell_3^m) \,]\!\!\} \;\;]\!\!\} \quad \doteq \\ \{\!\![& \underbrace{\{\!\![\, \texttt{false}, \texttt{true} \,]\!\!\}, \dots, \{\!\![\, \texttt{false}, \texttt{true} \,]\!\!\}}_{n}, \\ & \underbrace{\{\!\![\, \texttt{true}, A_1, B_1 \,]\!\!\}, \dots, \{\!\![\, \texttt{true}, A_m, B_m \,]\!\!\}}_{m} \;\;]\!\!\} \end{array}$$

(remember that k-element bags can only unify with k-element bags).

4.1.2 NP-hardness of compact-list unification

Any instance of 3-SAT can be mapped into the conjunction of unification problems

$$\begin{array}{rcl} [\![\, \mathtt{false}, X_i, \mathtt{false}, Y_i, \mathtt{false} \,]\!] & \doteq & [\![\, \mathtt{false}, \mathtt{true}, \mathtt{false} \,]\!] \\ [\![\, \mathtt{false}, f(\ell_1^j), f(\ell_2^j), f(\ell_3^j) \,]\!] & \doteq & [\![\, \mathtt{false}, \mathtt{true}, A_j, B_j \,]\!] \end{array}$$

for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$. Notice how an adequate number of alternations of `false` and `true` allows to force the desired matching. Equivalently, it can be reduced to the unique unification problem

$$\begin{array}{l} [\![\ [\![\, \mathtt{false}, X_1, \mathtt{false}, Y_1, \mathtt{false} \,]\!], \ldots, [\![\, \mathtt{false}, X_n, \mathtt{false}, Y_n, \mathtt{false} \,]\!], \\ \quad [\![\, \underline{2}, \mathtt{false}, f(\ell_1^1), f(\ell_2^1), f(\ell_3^1) \,]\!], \\ \quad \ldots, \\ \quad [\![\, \underline{1+m}, \mathtt{false}, f(\ell_1^m), f(\ell_2^m), f(\ell_3^m) \,]\!]\ ]\!] \ \doteq \\ [\![\ [\![\, \mathtt{false}, \mathtt{true}, \mathtt{false} \,]\!], \\ \quad [\![\, \underline{2}, \mathtt{false}, \mathtt{true}, A_1, B_1 \,]\!], \ldots, [\![\, \underline{1+m}, \mathtt{false}, \mathtt{true}, A_m, B_m \,]\!]\ ]\!]\,, \end{array}$$

where $\underline{k}$ stands for the term $\underbrace{[\![\cdots [\![}_{k}\ ]\!] \cdots]\!]$. Such numeral elements are introduced to ensure the correct matching.

4.1.3 NP-hardness of set unification

Any instance of 3-SAT can be mapped into the conjunction of set unification problems

$$\begin{array}{rcll} \{X_i, Y_i\} & \doteq & \{\mathtt{false}, \mathtt{true}\} & i = 1, \ldots, n \\ \{\mathtt{false}, f(\ell_1^j), f(\ell_2^j), f(\ell_3^j)\} & \doteq & \{\mathtt{false}, \mathtt{true}\} & j = 1, \ldots, m \end{array}$$

or, equivalently, into the unique unification problem

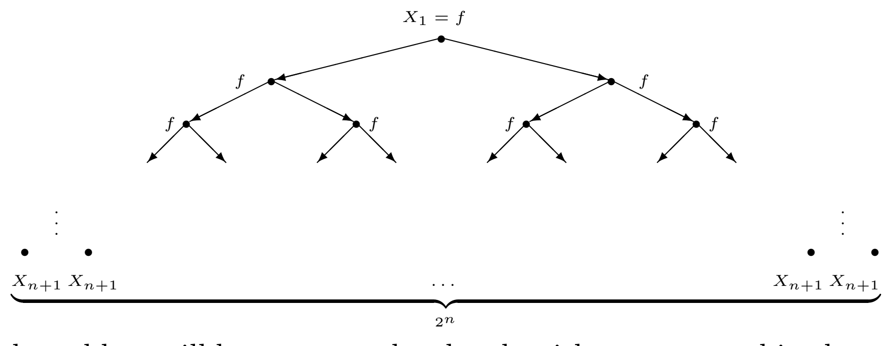
$$\begin{array}{ll} \{ & \{X_1, Y_1\}, \ldots, \{X_n, Y_n\}, \\ & \{\mathtt{false}, f(\ell_1^1), f(\ell_2^1), f(\ell_3^1)\}, \\ & \ldots, \\ & \{\mathtt{false}, f(\ell_1^m), f(\ell_2^m), f(\ell_3^m)\}\ \} \ \doteq \\ \{ & \{\mathtt{false}, \mathtt{true}\}\ \} \end{array}$$

For any of the three presented reductions, it is a matter of routine to verify that the restriction to U of any solution to Φ is an affirmative answer for $\langle C, U\rangle$. On the other hand, for any solution to $\langle C, U\rangle$ there exists a solution to the corresponding unification problem Φ extending it.

An alternative reduction for the set case is provided in [59].

4.2 À la Robinson: the well-founded case

The unification algorithms presented in this section are based on a rewriting technique firstly presented by Jacques Herbrand in his famous P.h.D. thesis (cf. [49]). The Herbrand algorithm was retaken and made famous by J. A. Robinson (cf. [95]) for its use in the implementation of the resolution algorithm. More in general, Although easy to understand and to implement, such algorithms share the unpleasant property to hidden an exponential space requirement due to the explicit application of substitution. For instance, when the input system is of the form $X_1 \doteq f(X_2, X_2), X_2 \doteq f(X_3, X_3), \ldots, X_n \doteq f(X_{n+1}, X_{n+1})$, the computed substitution for the variable X_1 is represented by the following diagram:



Such problem will be overcome by the algorithms presented in the next section § 4.3, following ideas and techniques from [92, 77]. Nevertheless, it is important to deeply study algorithms in the style of Robinson both for the easiness to implement them using meta-interpreters of Prolog and for the fact that they represent an important starting point for future optimizations.

Every unification algorithm that will be presented can be conceptually divided into two parts. A standard part consisting of actions already treated in Robinson's Algorithm, such as substitution application, (standard) term decomposition, etc. A part dealing with set-set (list-list, bag-bag, compact-list–compact-list) equations of the form $\{t_1 \mid s_1\} \doteq \{t_2 \mid s_2\}$. Such equations will be re-written accordingly with the corresponding equality principle: axiom (F_1) for lists, axioms (E_k^m), (E_k^c), and (E_k^s), for bags, compact lists, and sets, respectively.

The following definitions will be useful in the rest of the section.

Definition 4.1 *An equation set $\mathcal{E}$ is said to be* Solved *if it has the form $\{X_1 \doteq t_1, \ldots, X_n \doteq t_n\}$ and the X_i's are distinct variables which do not occur in terms r.h.s. of any equation of $\mathcal{E}$. $X_1, \ldots, X_n$ are said to be* Eliminable. *Similarly, any variable X occurring in $\mathcal{E}$ (not necessarily solved) only as l.h.s. of one equation is said to be* Eliminable.

Definition 4.2 *The* Size *of a term t is recursively defined as follows:*

$$\begin{cases} \mathit{size}(X) &= 0 \\ \mathit{size}(f(t_1, \ldots, t_n)) &= 1 + \sum_{i=1}^{n} \mathit{size}(t_i) \,. \end{cases}$$

In the algorithms that will follow, adopting standard Prolog syntax, identifiers beginning with capital letters will denote variables, (in particular N will denote a fresh variable, generated dynamically by the algorithm), identifiers beginning with small letters will denote functional symbols. s, t, s_i, t_i, etc. will stand for generic terms.

4.2.1 Hybrid lists unification

The unification algorithm for hybrid lists is basically the same algorithm presented by Herbrand in his famous thesis ([49]) and, later, used by Robinson to implement the resolution algorithm ([95]).

The aim of the algorithm is to return a solved form system $\mathcal{E}'$ equivalent, in the theory `WF_lists` (the well founded theory of hybrid lists presented in § 3.2.1) to the input system $\mathcal{E}$. The form of a solved form system allows to easily obtain a substitution (the—unique—*most general unifier*—cf., for instance, [73, 67]). As already said, an exponential space (hence time in a sequential implementation) can be necessary to

Unify_lists($\mathcal{E}$):

$$\begin{array}{lrcl}
(1) & X \doteq X \wedge \mathcal{E} & \mapsto & \mathcal{E} \\
(2) & \left.\begin{array}{r} t \doteq X \wedge \mathcal{E} \\ t \text{ is not a variable} \end{array}\right\} & \mapsto & X \doteq t \wedge \mathcal{E} \\
(3) & \left.\begin{array}{r} X \doteq t \wedge \mathcal{E} \\ X \text{ does not occur in } t \\ X \text{ occurs in } \mathcal{E} \end{array}\right\} & \mapsto & \mathcal{E}[X/t] \wedge X \doteq t \\
(4) & \left.\begin{array}{r} X \doteq t \wedge \mathcal{E} \\ X \not\equiv t \text{ and } X \text{ occurs in } t \end{array}\right\} & \mapsto & \texttt{fail} \\
(5) & \left.\begin{array}{r} f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n) \wedge \mathcal{E} \\ f \text{ different from } g \end{array}\right\} & \mapsto & \texttt{fail} \\
(6) & f(s_1, \ldots, s_m) \doteq f(t_1, \ldots, t_m) \wedge \mathcal{E} & \mapsto & \\
 & s_1 \doteq t_1 \wedge \ldots \wedge s_m \doteq t_m \wedge \mathcal{E} & &
\end{array}$$

Figure 4.1: List unification algorithm *à la* Robinson

perform such rewriting. This bad behavior marks all unification algorithm presented in § 4.2.2, § 4.2.3, and § 4.2.4; however they can easily be implemented inside a logic programming environment.

The variable $\mathcal{E}$ is viewed as a conjunction of equations; to this aim, it can be considered as a *set*. From an operational point of view, it can be useful to see it as a multi set; however, all the results can be proved exactly in the same way.

In the algorithm of Fog. 4.1 f and g range over Σ; action (4) performs the so-called *occur-check*. It checks the well-foundedness of the unique most general solution solution to the system.

The termination of the algorithm is easy to prove:

Theorem 4.3 (Termination) Unify_lists($\mathcal{E}$) *always terminates.*

Proof. We prove that any non-failing action cause the decreasing of an adequate complexity measure.

- Action (1) decreases the number of equations of $\mathcal{E}$ (and does not increase anything else).

- Action (2) decreases the sum of the *size* of the l.h.s. term of the equations of $\mathcal{E}$.
- Action (3) decreases the number of non-eliminable variables of $\mathcal{E}$ (observe that it can increase the *size* of the terms containing X occurring in some equation of $\mathcal{E}$).
- Action (6) decreases the sum of the *size* of the l.h.s. of the equations of $\mathcal{E}$ (and also of the r.h.s.; moreover, observe that it can increase the number of equations of $\mathcal{E}$).

A complexity measure based on the triple of non-negative integers $\langle A, B, C \rangle$, where

- A is the number of non-eliminable variables of $\mathcal{E}$,
- $B = \sum_{\ell \doteq r \, in \, \mathcal{E}} size(\ell)$, and
- C is the number of equations in $\mathcal{E}$,

lexicographically ordered, decreases at each non failing-action. The termination follows from the well-foundedness of the lexicographical ordering of tuples of non-negative integers.

4.3 □

The correctness and completeness of the algorithm with respect to the corresponding theory presented in § 3.2.1 is proved in the following

Theorem 4.4 (Soundness and Completeness) *Let $\mathcal{E}$ be an equation system, and $\mathcal{E}'$ be the equation system resulting from the computation of* $\mathsf{Unify\_lists}(\mathcal{E})$ *(we assume that when the computation fails, then $\mathcal{E}' = \texttt{false}$). Then* $\texttt{WF\_lists} \vdash \mathcal{E} \leftrightarrow \mathcal{E}'$*.*

Proof. We prove the correctness result for each action. The termination of the algorithm, guaranteed by Theorem 4.3 ensure that the result holds globally.

Correctness of actions (1), (2), and (3) are ensured by equality axioms $(\doteq_1)$ and $(\doteq_2)$.

Correctness of failing actions (5) and (6) follow from freeness axioms (F_3) and (F_2), respectively.

Completeness of action (6) is justified by (F_1); its correctness is ensured by equality.

4.4 □

4.2.2 Hybrid multi-sets unification

The basic difference between the theories `WF_lists` and `WF_bags` is that freeness axiom (F_1) (related to action (6) of the unification algorithm Unify_lists), does not hold anymore when dealing with bags (cf. § 3.2.2). Axiom (E_k^m), which takes its role for bags, introduces a source of non-determinism.

Therefore, the aim of a unification algorithm for hybrid bags (the same will hold also for compact lists and sets), is to return, non-deterministically, a (finite) number of solved form systems $\mathcal{E}_1, \ldots, \mathcal{E}_k$, all together equivalent to the input system $\mathcal{E}$ (see Theorem 4.6), or to return `fail`, when $\mathcal{E}$ is unsatisfiable in `WF_bags`.

We first introduce a naive fully non-deterministic algorithm naive_-Unify_bags. We will show that it does not terminate when the input system is of a certain form. Then we will introduce a little of determinism, in order to guarantee termination.

The first five actions of naive_Unify_bags are the same as in the algorithm for hybrid lists presented in the previous section. We need to restrict action (6) for non-bag terms only, and to introduce a bag-bag case (7). This case introduces a source of *don't know* non-determinism; at run time both the alternatives (*i*) and (*ii*) must be exploited.

```
naive_Unify_bags($\mathcal{E}$):
  (1)—(5)  as in Unify_lists
  (6)      $f(s_1,\ldots,s_m) \doteq f(t_1,\ldots,t_m) \wedge \mathcal{E}$ }
           $f \in \Sigma \setminus \{\{\!\{\cdot\,|\,\cdot\}\!\}\}$               }      $\mapsto$
                $s_1 \doteq t_1 \wedge \ldots \wedge s_m \doteq t_m \wedge \mathcal{E}$
  (7)      $\{\!\{t\,|\,s\}\!\} \doteq \{\!\{t'\,|\,s'\}\!\} \wedge \mathcal{E}$                 $\mapsto$
                 $(i)$   $t \doteq t' \wedge s \doteq s' \wedge \mathcal{E}$
                 $(ii)$  $s \doteq \{\!\{t'\,|\,N\}\!\} \wedge \{\!\{t\,|\,N\}\!\} \doteq s' \wedge \mathcal{E}$
```

Although sound and complete with respect to the theory `WF_bags`, for some initial $\mathcal{E}$ there is some infinite sequence of transformation steps, applying actions (1)–(7); examples of inputs leading to non-termination are the following:

1. $\{\!\!\{ T \mid S \}\!\!\} \doteq \{\!\!\{ T' \mid S \}\!\!\} \overset{7(ii)}{\mapsto}$
   $S \doteq \{\!\!\{ T' \mid N \}\!\!\} \wedge \{\!\!\{ T \mid N \}\!\!\} \doteq S \overset{3}{\mapsto}$
   $S \doteq \{\!\!\{ T' \mid N \}\!\!\} \wedge \{\!\!\{ T \mid N \}\!\!\} \doteq \{\!\!\{ T' \mid N \}\!\!\}$.
   The last system contains a subsystem equal—modulo variable renaming—to the one we started from.

2. $\{\!\!\{ T_1 \mid S_1 \}\!\!\} \doteq \{\!\!\{ T_2 \mid S_2 \}\!\!\} \wedge \{\!\!\{ T_3 \mid S_2 \}\!\!\} \doteq \{\!\!\{ T_4 \mid S_1 \}\!\!\} \overset{7(ii)}{\mapsto}$
   $S_1 \doteq \{\!\!\{ T_2 \mid N_1 \}\!\!\} \wedge \{\!\!\{ T_1 \mid N_1 \}\!\!\} \doteq S_2 \wedge \{\!\!\{ T_3 \mid S_2 \}\!\!\} \doteq \{\!\!\{ T_4 \mid S_1 \}\!\!\} \overset{7(ii)}{\mapsto}$
   $S_1 \doteq \{\!\!\{ T_2 \mid N_1 \}\!\!\} \wedge \{\!\!\{ T_1 \mid N_1 \}\!\!\} \doteq S_2 \wedge$
   $\quad S_2 \doteq \{\!\!\{ T_4 \mid N_2 \}\!\!\} \wedge \{\!\!\{ T_3 \mid N_2 \}\!\!\} \doteq S_1 \overset{2-3-3}{\mapsto}$
   $S_1 \doteq \{\!\!\{ T_2 \mid N_1 \}\!\!\} \wedge S_2 \doteq \{\!\!\{ T_1 \mid N_1 \}\!\!\} \wedge$
   $\quad \{\!\!\{ T_1 \mid N_1 \}\!\!\} \doteq \{\!\!\{ T_4 \mid N_2 \}\!\!\} \wedge \{\!\!\{ T_3 \mid N_2 \}\!\!\} \doteq \{\!\!\{ T_2 \mid N_1 \}\!\!\}$.

3. More generally, for any situation of the form
   $\{\!\!\{ \cdots \mid S_1 \}\!\!\} \doteq \{\!\!\{ \cdots \mid S_2 \}\!\!\} \wedge \{\!\!\{ \cdots \mid S_2 \}\!\!\} \doteq \{\!\!\{ \cdots \mid S_3 \}\!\!\} \wedge \ldots \wedge$
   $\quad \{\!\!\{ \cdots \mid S_n \}\!\!\} \doteq \{\!\!\{ \cdots \mid S_1 \}\!\!\}$,
   it is easy to find a non-deterministic sequence of actions leading to non-termination.

The (base) situation described in the above example (1) can be easily handled as special case: let tail and de_tail be the functions defined below (X will denote a generic variable):

$$\begin{cases} \mathsf{tail}(\{\!\!\{\,\}\!\!\}) & = & \{\!\!\{\,\}\!\!\} \\ \mathsf{tail}(X) & = & X \\ \mathsf{tail}(\{\!\!\{ t \mid s \}\!\!\}) & = & \mathsf{tail}(s) \end{cases} \qquad \begin{cases} \mathsf{de\_tail}(X) & = & \{\!\!\{\,\}\!\!\} \\ \mathsf{de\_tail}(\{\!\!\{ t \mid s \}\!\!\}) & = & \{\!\!\{ t \mid \mathsf{de\_tail}(s) \}\!\!\} \end{cases};$$

then *action* (7) of the above algorithm can be split into two sub-actions. If tail(s) and tail(s') are not the same variable then perform action (7). Otherwise replace $\{\!\!\{ t \mid s \}\!\!\} \doteq \{\!\!\{ t' \mid s' \}\!\!\}$ with:

$\mathsf{de\_tail}(\{\!\!\{ t \mid s \}\!\!\}) \doteq \mathsf{de\_tail}(\{\!\!\{ t' \mid s' \}\!\!\})$.[1]

[1] To be precise, in this way, a variable X occurring in a system only as *tail* of the two bags analyzed, disappears from the system. This is not a problem for correctness and completeness, since the above equation does not force any constraint for X.

The reason for non-termination is that the unification algorithm looks for all substitutions for S such that $\{\!\{ T \,|\, S \}\!\} \doteq \{\!\{ T' \,|\, S \}\!\}$, i.e. when S is left 'free', when it contains at least one element, two elements, and so on. clearly, the first solution is more general than the others. For the base case the remedy has already been described. To overcome counter-examples (2) and (3) action (7) is modified and a particular control is imposed to the application of the actions adopting a stack data structure. This will be sufficient to guarantee that when an equation $\{\!\{ s_1, \ldots, s_m \,|\, S \}\!\} \doteq \{\!\{ t_1, \ldots, t_n \,|\, S' \}\!\}$ is encountered, the sequence of actions executed is such that a conjunction of equations of the form $s_{i_1} \doteq t_{j_1}, \ldots, s_{i_k} \doteq t_{j_k}$, plus two equations of the form $S \doteq \{\!\{ t_{j_{k+1}}, \ldots, t_{j_n} \,|\, N \}\!\}, S' \doteq \{\!\{ s_{i_{k+1}}, \ldots, s_{i_m} \,|\, N \}\!\}$ or one equation of the form $S \doteq \{\!\{ t_{j_{k+1}}, \ldots, t_{j_n} \,|\, S' \}\!\}$ or one of the form $S' \doteq \{\!\{ s_{i_{k+1}}, \ldots, s_{i_m} \,|\, S \}\!\}$ (cf. Lemma 4.7) is returned. After that, applying the substitutions related to variables S and S', even if a new variable N is introduced into the system, one or both variables S and S' become *eliminable* (see Def. 4.1), i.e., in a sense, disappear from the system (this will be better explained in the proof of Theorem 4.5).

For instance, counter-example 2 will be overcome:

$$
\begin{array}{ll}
\{\!\{ T_1 \,|\, S_1 \}\!\} \doteq \{\!\{ T_2 \,|\, S_2 \}\!\} \wedge \{\!\{ T_3 \,|\, S_2 \}\!\} \doteq \{\!\{ T_4 \,|\, S_1 & \overset{7(ii)}{\mapsto} \\
S_1 \doteq \{\!\{ T_2 \,|\, N_1 \}\!\} \wedge \{\!\{ T_1 \,|\, N_1 \}\!\} \doteq S_2 \wedge \{\!\{ T_3 \,|\, S_2 \}\!\} \doteq \{\!\{ T_4 \,|\, S_1 \}\!\} & \overset{2-3-3}{\mapsto} \\
S_1 \doteq \{\!\{ T_2 \,|\, N_1 \}\!\} \wedge S_2 \doteq \{\!\{ T_1 \,|\, N_1 \}\!\} \wedge \{\!\{ T_3, T_1 \,|\, N_1 \}\!\} \doteq \{\!\{ T_4, T_2 \,|\, N_1 \}\!\} \,. &
\end{array}
$$

The last equation will be replaced by $\{\!\{ T_3, T_1 \}\!\} \doteq \{\!\{ T_4, T_2 \}\!\}$, avoiding the loop.

This circle of ideas is summarized by the algorithm in Fig. 4.2.

Theorem 4.5 (Termination) Unify_bags *always terminates, for any input system $\mathcal{E}$.*

Proof. We define a complexity measure $\langle A_{\mathcal{E}}, B_{\mathcal{E}}, C_{\mathcal{E}} \rangle$, where $A_{\mathcal{E}}, B_{\mathcal{E}}, C_{\mathcal{E}}$ are non-negative integers depending on the value of $\mathcal{E}$. We will show that each non-failing action of Unify_bags decreases the value of $\langle A_{\mathcal{E}}, B_{\mathcal{E}}, C_{\mathcal{E}} \rangle$ with respect to the lexicographical ordering. Since the lexicographical ordering on tuples of non-negative integers is a well-ordering, this is sufficient to prove the termination of the algorithm.

```
Unify_bags($\mathcal{E}$);
stack := empty;
while $\mathcal{E}$ is not in solved form or not empty(stack) do
  begin
    while not empty(stack) do
      begin
        $e$ := pop(stack)
        if $e$ is in solved form with respect to ($\mathcal{E}$ and stack)
        then $\mathcal{E} := e \wedge \mathcal{E}$
        else action
      end
    select arbitrarily from $\mathcal{E}$ an equation $e$ not in solved form; action
  end.

action:
let $\mathcal{E}'$ be $\mathcal{E} \setminus \{e\}$;
  case $e$ of
(1)  $X \doteq X$  ↦  $\mathcal{E} := \mathcal{E}'$
(2)  $t \doteq X$, $t$ is not a variable  ↦  $\mathcal{E} := X \doteq t \wedge \mathcal{E}'$
(3)  $X \doteq t$, $X$ does not occur in $t$, $X$ occurs in $\mathcal{E}'$  ↦  $\mathcal{E} := \mathcal{E}'[X/t] \wedge X \doteq t$;
                                                                     stack := stack$[X/t]$
(4)  $X \doteq t$, $X \not\equiv t$ and $X$ occurs in $t$  ↦  fail
(5)  $f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_m)$  ↦  fail
(6)  $f(s_1, \ldots, s_m) \doteq f(t_1, \ldots, t_m)$, $f \in \Sigma \setminus \{\{\!\!\{\cdot \mid \cdot\}\!\!\}\}$  ↦
         $\mathcal{E} := s_1 \doteq t_1 \wedge \ldots \wedge s_m \doteq t_m \wedge \mathcal{E}'$
(7)  $\{\!\!\{ t \mid s \}\!\!\} \doteq \{\!\!\{ t' \mid s' \}\!\!\}$, tail($s$) and tail($s'$) are the same variable  ↦
         $\mathcal{E} := \mathcal{E}'$; push(de_tail($\{\!\!\{ t \mid s \}\!\!\}$) $\doteq$ de_tail($\{\!\!\{ t' \mid s' \}\!\!\}$), stack);
(8)  $\{\!\!\{ t \mid s \}\!\!\} \doteq \{\!\!\{ t' \mid s' \}\!\!\}$, tail($s$) and tail($s'$) are not the same variable  ↦
         $(i)$   $\mathcal{E} := t \doteq t' \wedge \mathcal{E}'$;
         $(ii)$  push($s \doteq \{\!\!\{ t' \mid N \}\!\!\}$, stack);
                 push($\{\!\!\{ t \mid N \}\!\!\} \doteq s'$, stack);
```

Figure 4.2: Multi-set unification algorithm *à la* Robinson

- $A_{\mathcal{E}}$ is the number of non-eliminable variables of $\mathcal{E}$ (see Def. 4.1).
- $B_{\mathcal{E}}$ is the *left size* of $\mathcal{E}$, i.e. $\sum_{(\ell \doteq r)\, in\, \mathcal{E}} size(\ell)$ (see Def. 4.2).
- $C_{\mathcal{E}}$ is the number of equations of $\mathcal{E}$.

A *phase* of the computation of the algorithm is defined to be the

- a single action $(*)$ if the stack is empty and $(*)$ does not modify the variable stack;
- the (finite) sequence of actions that begins with action $8(ii)$ and ends with an action distinct from $8(ii)$ and fired by the equation selected from the stack leaving it empty.

We will analize, for each non-failing phase, the behavior of the selected measure of complexity:

1: $A_{\mathcal{E}}$ and $B_{\mathcal{E}}$ are left unchanged; $C_{\mathcal{E}}$ decreases;

2, 5: $A_{\mathcal{E}}$ is left unchanged; $B_{\mathcal{E}}$ decreases;

3: $A_{\mathcal{E}}$ decreases;

7: c.f. the analysis of action (8) when tail(s) and tail(s') are not variables (i.e. they are both $\{\!\!\{\ \}\!\!\}$);

(8): Assume tail(s) and tail(s') are not variables. The data structure stack ensure that a conjunction of equations between elements is returned, together with an equation regarding the kernels of the two bags. When the latter does not force an immediate failure due to subsequent action (5), a new conjunction of equations between smaller terms is introduced by action (6). $A_{\mathcal{E}}$ is left unchanged, while $B_{\mathcal{E}}$ decreases.

Assume tail(s) is the variable S and tail(s') is a kernel $f(\ldots)$. A substitution of the form $[S/\{\!\!\{\, r_1, \ldots, r_n \,|\, f(\ldots) \,\}\!\!\}]$, where n is the difference between the number of elements of the rightmost bag and the number of elements of the leftmost one when S is replaced by $\{\!\!\{\ \}\!\!\}$. Thanks to the substitution application, $A_{\mathcal{E}}$ decreases.

The proof for the symmetrical case (namely when tail(s') is the variable S and tail(s) is a kernel $f(\ldots)$) is similar.

Assume tail(s) is the variable S and tail(s') is the variable S' (the distinction between actions (7) and (8) ensures that S and S' are distinct). As shown in Lemma 4.7, a new (and not solved) variable N can be introduced. If this is the case, however, a substitution of the form $[S/\{\!\![\, r_1, \ldots, r_n \,|\, N \,]\!\!\}, S'/\{\!\![\, \ell_1, \ldots, \ell_m \,|\, N \,]\!\!\}]$ is applied: $A_{\mathcal{E}}$ decreases. If, conversely, a new variable is not introduced, then a substitution of one of the form $[S/\{\!\![\, r_1, \ldots, r_n \,|\, S' \,]\!\!\}]$ or of the form $[S'/\{\!\![\, \ell_1, \ldots, \ell_m \,|\, S \,]\!\!\}]$ is applied, decreasing $A_{\mathcal{E}}$.

4.5 □

Theorem 4.6 (Soundness and Completeness) *Let $\mathcal{E}$ be an equation system, $\mathcal{E}_1, \ldots, \mathcal{E}_h$ be the equation systems non-deterministically resulting from the computation of* Unify_bags$(\mathcal{E})$*. Let $N_1, \ldots, N_k$ be the variables occurring in $\mathcal{E}_1, \ldots, \mathcal{E}_h$ but not in $\mathcal{E}$; then* WF_bags $\vdash \mathcal{E} \leftrightarrow \exists N_1, \ldots, N_k \bigvee_{i=1}^{h} \mathcal{E}_i$.

Proof. As for the proof of Theorem 4.4, it is sufficient to prove the claim for each single action.

Actions (1)–(6) are the same as in algorithm Unify_lists whose correctness has been proved in Theorem 4.4.

Since two bags are equal if and only if they contains the same number of occurrences of each element, then the problem

$$\{\!\![\, s_1, \ldots, s_m \,|\, X \,]\!\!\} \doteq \{\!\![\, t_1, \ldots, t_n \,|\, X \,]\!\!\}$$

is perfectly equivalent to the problem

$$\{\!\![\, s_1, \ldots, s_m \,]\!\!\} \doteq \{\!\![\, t_1, \ldots, t_n \,]\!\!\}.$$

Soundness and completeness of action (8) follows from the fact that action (7) of algorithm naive_Unify_bags is exactly axiom (E_k^m).

4.6 □

Given an equational theory $\mathcal{T}$, for any satisfiable Herbrand system $\mathcal{E}$ involving terms from $\tau(\Sigma \cup \mathcal{V})$, a unification algorithm should be able to cover through non-determinism *each element* of a complete set of $\mathcal{T}$-unifiers of $\mathcal{E}$ (c.f. § 2.2). From Theorem 4.6 we can infer that $\{\mathcal{E}_1, \ldots, \mathcal{E}_h\}$ is, in fact, a complete set of (E_1)-unifiers of $\mathcal{E}$. However, nothing can be stated about the minimality properties of $\mathcal{E}_1, \ldots, \mathcal{E}_h$, namely whether they are all independent or not.

In general, a valid criterion to compare two unification algorithms is the analysis of the length of the *list* of solutions computed by them. (The word 'list' is used here to reflect the fact that, if a unification algorithm computes exactly the minimal complete set of unifiers $\mu\bigcup_{\mathcal{T}}(\mathcal{E})$ but some solution is returned more than once, then it cannot be considered minimal).

In the rest of this section we will analyze the behavior, with respect to the minimality, of the execution of the unification on some (significant) sample problems. First we present the following general result.

Lemma 4.7 *Let $A_1, \ldots, A_m, B_1, \ldots, B_n, S, S'$ $(m \leq n)$ be pairwise distinct variables. Any solution (even non ground) representable with finite trees to the unification problem*

$$\{\!\!\{ A_1, \ldots, A_m \,|\, S \}\!\!\} \doteq \{\!\!\{ B_1, \ldots, B_n \,|\, S' \}\!\!\}$$

is of the form

$$\begin{array}{ll} A_{i_1} \doteq B_{j_1}, \ldots, A_{i_k} \doteq B_{j_k}, & \\ S \doteq \{\!\!\{ B_{j_{k+1}}, \ldots, B_{j_n} \,|\, N \}\!\!\} & (\textit{if } n = k \textit{ then } N = S) \\ S' \doteq \{\!\!\{ A_{i_{k+1}}, \ldots, A_{i_m} \,|\, N \}\!\!\} & (\textit{if } m = k \textit{ then } N = S') \end{array}$$

where

- *$i : \{1, \ldots, m\} \longrightarrow \{1, \ldots, m\}$ and $j : \{1, \ldots, n\} \longrightarrow \{1, \ldots, n\}$ are two permutations;*
- *k is an integer $0 \leq k \leq \min\{m, n\}$.*

Proof. It is easy to see that each (solved form) system of the form above is a solution to the unification problem at hand in a bag theory. For the converse direction, let γ be a ground unifier of

$$\{\!\!\{ A_1, \ldots, A_m \,|\, S \}\!\!\} \doteq \{\!\!\{ B_1, \ldots, B_n \,|\, S' \}\!\!\}$$

and let a_i, $i = 1, \dots, m$, and b_j, $j = 1, \dots, n$, be $A_i\gamma$ and $B_j\gamma$, respectively.
Since we are interested in finite solutions only, $S\gamma$ and $S'\gamma$ should have the form $\{\!\![\, a_{m+1}, \dots, a_h \,]\!\!\}$ and $\{\!\![\, b_{n+1}, \dots, b_k \,]\!\!\}$ ($k \geq \max\{m, n\}$), respectively.

From a simple adaptation to Theorem 3.10for the hybrid case it follows that $h = k$ and, moreover, a permutation $\pi : \{1, \dots, k\} \longrightarrow \{1, \dots, k\}$ such that $a_{\pi_i} \doteq b_i$, for $i = 1, \dots, k$, exists.
It is immediate to see that γ is an instance of one of the unifiers described in the statement.

4.7 □

Thus, any solution to the problem can be obtained selecting a bag consisting of i elements from $\{\!\![\, A_1, \dots, A_m \,]\!\!\}$ and a bag consisting of i elements from $\{\!\![\, B_1, \dots, B_n \,]\!\!\}$. Then a bijection among them has to be chosen.

Corollary 4.8 *The bag unification problem*

$$\{\!\![\, A_1, \dots, A_m \mid S \,]\!\!\} \doteq \{\!\![\, B_1, \dots, B_n \mid S' \,]\!\!\}$$

(A_i's, B_j's, S, and S' pairwise distinct variables) admits exactly

$$\sum_{i=0}^{\min\{m,n\}} \binom{m}{i}\binom{n}{i} i!$$

independent solutions. □

With the following Lemma and Theorem we will prove the minimality of the presented bag unification algorithm with respect to the selected sample problem:

Lemma 4.9 *Let $A_1, \dots, A_m, B_1, \dots, B_n, S, S'$ be pairwise distinct variables. Then*

$\mathsf{Unify\_bags}(\{\!\![\, A_1, \dots, A_m \mid S \,]\!\!\} \doteq \{\!\![\, B_1, \dots, B_n \mid S' \,]\!\!\})$

returns exactly $f(m, n)$ solutions, where f is recursively defined as follows:

$$\begin{cases} f(m,0) & = & f(0,n) & = & 1 \\ f(m+1,n+1) & = & (m+1)f(m,n) + f(m+1,n) \end{cases}$$

Proof. If $m = 0$ or $n = 0$ the result is trivial. Since Unify_bags is simply a more deterministic version of naive_Unify_bags, and that the particular form of the input guarantees the termination of the latter algorithm on it, without loss of generality, we will prove the claim for the simplest algorithm.

In order to compute the value for $f(m+1, n+1)$, assume the problem is $\{\!\!\{\, A_0, \ldots, A_m \mid S \,\}\!\!\} \doteq \{\!\!\{\, B_0, \ldots, B_n \mid S' \,\}\!\!\}$. The unification algorithm performs, non-deterministically, one of the two sub-actions of action (7):

(i) $A_0 = B_0$ and $\{\!\!\{\, A_1, \ldots, A_m \mid S \,\}\!\!\} \doteq \{\!\!\{\, B_1, \ldots, B_n \mid S' \,\}\!\!\}$: since variables are all distinct, $f(m, n)$ solutions are returned.

(ii) $\{\!\!\{\, A_1, \ldots, A_m \mid S \,\}\!\!\} \doteq \{\!\!\{\, B_0 \mid N \,\}\!\!\}$ and $\{\!\!\{\, A_0 \mid N \,\}\!\!\} \doteq \{\!\!\{\, B_1, \ldots, B_n \mid S' \,\}\!\!\}$: following the termination strategy, the unification algorithm operates first on the former equation until it is reduced to a solved form system, and later on the latter.

The first computation reports a solution which, restricted to the initial variables, is of one of the two forms

$$\begin{array}{ll} (\circ) & A_i \doteq B_0 \wedge N \doteq \{\!\!\{\, A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_m \mid S \,\}\!\!\} \\ (\bullet) & S \doteq \{\!\!\{\, B_0 \mid N' \,\}\!\!\} \wedge N \doteq \{\!\!\{\, A_1, \ldots, A_m \mid N' \,\}\!\!\}\ . \end{array}$$

In particular, m solutions of the form $(\circ)$ and one of the form $(\bullet)$ are returned; applying the substitution for N, m subproblems returning $f(m, n)$ solutions and one returning $f(m+1, n)$ are generated.

Hence, we have:

$$\begin{array}{rcl} f(m+1, n+1) & = & \underbrace{f(m,n)}_{\text{from } (i)} + \underbrace{m \cdot f(m,n)}_{\text{from } (ii)\circ} + \underbrace{\cdot f(m+1,n)}_{\text{from } (ii)\bullet} \\ & = & (m+1) \cdot f(m,n) + f(m+1,n) \end{array}$$

4.9 □

Theorem 4.10 *If f is the function defined in the statement of Lemma 4.9, then*

$$f(m,n) = \sum_{i=0}^{\min\{m,n\}} \binom{m}{i}\binom{n}{i}\, i!\,.$$

Proof. When $m = 0$ or $n = 0$ the result is trivial. We conclude the proof by induction. Assume that $m < n$ (if $m \geq n$ the proof is similar):

$$\begin{aligned}
f(m+1,n+1) &= \textstyle\sum_{i=0}^{m+1} \binom{m+1}{i}\binom{n+1}{i}\, i! \\
&\quad \text{(since } \tbinom{m+1}{0}\tbinom{n+1}{0}0! = 1\text{)} \\
&= \textstyle\sum_{i=0}^{m} \binom{m+1}{i+1}\binom{n+1}{i+1}\,(i+1)! + 1 \\
&\quad \text{(since } \tbinom{n+1}{i+1} = \tbinom{n}{i} + \tbinom{n}{i+1}\text{)} \\
&= \textstyle\sum_{i=0}^{m} \binom{m+1}{i+1}\left(\binom{n}{i} + \binom{n}{i+1}\right)\,(i+1)! + 1 \\
&= \textstyle\sum_{i=0}^{m} \binom{m+1}{i+1}\binom{n}{i}(i+1)! + \\
&\quad \textstyle\sum_{i=0}^{m} \binom{m+1}{i+1}\binom{n}{i+1}(i+1)! + 1
\end{aligned}$$

Since $\binom{m+1}{i+1} = \frac{m+1}{i+1}\binom{m}{i}$ and $\binom{m+1}{0}\binom{n}{0}1! = 1$, then

$$\begin{aligned}
f(m+1,n+1) &= \textstyle\sum_{i=0}^{m} \frac{m+1}{i+1}\binom{m}{i}\binom{n}{i}(i+1)i! + \\
&\quad \textstyle\sum_{i=0}^{m+1} \binom{m+1}{i+1}\binom{n}{i+1}(i+1)! + 1 \\
&= (m+1)f(m,n) + f(m+1,n)
\end{aligned}$$

4.10 □

Nevertheless, Unify_bags is not minimal for all instances of the unification problem. For instance, it returns three solutions to

$$\{\!\!\{\, A, A \mid S \,\}\!\!\} \doteq \{\!\!\{\, A \mid S' \,\}\!\!\}$$

whereas the complete set of unifiers can be described by the unique solution $S' = \{\!\!\{\, A \mid S \,\}\!\!\}$.

Minimality can be reached also for this and other cases, provided the algorithm is modified to deal with various special cases. In this thesis, however, we only analize optimization techniques of this kind for set unification (see § 4.4).

4.2.3 Hybrid compact-lists unification

The unification algorithm for hybrid compact-lists should differ from the lists one both for the fact that axiom (F_1) does not hold for compact lists, and, as shown in § 3.1.3, that axiom (F_3) must be modified accordingly to the semantics to the compact-list constructor $[\![\, \cdot \,|\, \cdot \,]\!]$.

Similarly to the bag case, the former difference is handled by modifying action (6) of Unify_lists restricting it to non compact-list case only and introducing a new non-deterministic action reflecting the semantics of axiom (E_k^c). The latter difference is reflected into the occur check performed by action (4)—taking into account the freeness requirement of axiom schema (F_3^c)—of the algorithm Unify_clists described in Fig. 4.3.

We only give here the statement of the termination theorem (for future references). Its proof will follow by the proof of the corresponding one for sets (Theorem 4.16) presented in the next section.

Theorem 4.11 (Termination) *For any input system $\mathcal{E}$,* Unify_clists *always terminates, no matter what non-deterministic sequence of choices is made.*

Theorem 4.12 (Soundness and Completeness) *Let $\mathcal{E}$ be an equation system, $\mathcal{E}_1, \dots, \mathcal{E}_h$ the equation systems non-deterministically resulting from the computation of* Unify_clists($\mathcal{E}$)*. Let $N_1, \dots, N_k$ be the variables occurring in $\mathcal{E}_1, \dots, \mathcal{E}_h$ but not in $\mathcal{E}$; then* `WF_clists` $\vdash \mathcal{E} \leftrightarrow \exists N_1, \dots, N_k \bigvee_{i=1}^{h} \mathcal{E}_i$.

Proof. As for Theorem 4.4, it is sufficient to prove the claim for each single action application. By case analysis. Actions (1)–(3) and (6)–(7) are the same as in algorithm Unify_lists whose correctness has been proved in Theorem 4.4.

Correctness of actions (4) and (5) ensues from an intuitive extension of Lemma 3.21 to the hybrid case.

Action (8) reflects exactly axiom (E_k^c).

4.12 □

Theorem 4.12 proved also that $\{\mathcal{E}_1, \dots, \mathcal{E}_h\}$ is a complete set of (E_2)-unifiers for $\mathcal{E}$.

Let us give a look to the minimality properties of Unify_clists.

Lemma 4.13 *Let $A_1, \dots, A_m, B_1, \dots, B_n, S, S'$ be pairwise distinct variables. Then*

$$\mathsf{Unify\_clists}([\![\, A_1, \dots, A_m \,|\, S \,]\!] \doteq [\![\, B_1, \dots, B_n \,|\, S' \,]\!])$$

```
Unify_clists($\mathcal{E}$);
(1)  $X \doteq X \wedge \mathcal{E} \mapsto \mathcal{E}$
(2)  $t \doteq X \wedge \mathcal{E}$
     $t$ is not a variable
        $\mapsto X \doteq t \wedge \mathcal{E}$
(3)  $X \doteq t \wedge \mathcal{E}$
     $X$ does not occur in $t$
     $X$ occurs in $\mathcal{E}$
        $\mapsto \mathcal{E}[X/t] \wedge X \doteq t$
(4)  $X \doteq [\![ t_0, \ldots, t_n \,|\, X ]\!] \wedge \mathcal{E}$
     $X$ does not occur in $t_0, \ldots, t_n$
        $\mapsto$
     $(Y \doteq t_0 \wedge \ldots \wedge Y \doteq t_n \wedge \mathcal{E})[X/[\![ Y \,|\, Z ]\!]] \wedge X \doteq [\![ Y \,|\, Z ]\!]$
(5)  $X \doteq s \wedge \mathcal{E}$
     ($s$ is $f(t_0, \ldots, t_n)$, $f \not\equiv [\![ \cdot \,|\, \cdot ]\!]$
     and $X$ occurs in $s$) or
     ($s$ is $[\![ t_0, \ldots, t_n \,|\, t ]\!]$, $t \not\equiv [\![ \cdot \,|\, \cdot ]\!]$
     and ($X$ occurs in $t_i$, for some $0 \leq i \leq n$ or
     $t \not\equiv X$ and $X$ occurs in $t$))
        $\mapsto$ fail
(6)  $f(s_1, \ldots, s_m) \doteq g(t_1, \ldots, t_n) \wedge \mathcal{E}$
     $f$ different from $g$
        $\mapsto$ fail
(7)  $f(s_1, \ldots, s_m) \doteq f(t_1, \ldots, t_m) \wedge \mathcal{E}$
     $f \in \Sigma \{[\![ \cdot \,|\, \cdot ]\!]\}$
        $\mapsto$
     $s_1 \doteq t_1 \wedge \ldots \wedge s_m \doteq t_m \wedge \mathcal{E}$
(8)  $[\![ t \,|\, s ]\!] \doteq [\![ t' \,|\, s' ]\!] \wedge \mathcal{E} \mapsto$
     $(i)$    $t \doteq t' \wedge s \doteq s' \wedge \mathcal{E}$
     $(ii)$   $t \doteq t' \wedge s \doteq [\![ t' \,|\, s' ]\!] \wedge \mathcal{E}$
     $(iii)$  $t \doteq t' \wedge [\![ t \,|\, s ]\!] \doteq s' \wedge \mathcal{E}$
```

Figure 4.3: Compact-list unification algorithm *à la* Robinson

returns exactly $f(m,n)$ solutions, where f is recursively defined as follows:

$$\begin{cases} f(m,0) & = & f(0,n) & = & 1 \\ f(m+1,n+1) & = & f(m,n)+f(m+1,n)+f(m,n+1)\,. \end{cases}$$

Proof. If $m = 0$ or $n = 0$ the result is trivial. To compute the value for $f(m+1,n+1)$, assume the problem is $\llbracket A_0,\ldots,A_m \,|\, S \rrbracket \doteq \llbracket B_0,\ldots,B_n \,|\, S' \rrbracket$. The unification algorithm performs, non-deterministically, one of the sub-actions

(i): exploring $\llbracket A_1,\ldots,A_m \,|\, S \rrbracket \doteq \llbracket B_1,\ldots,B_n \,|\, S' \rrbracket$,

(ii): analyzing $\llbracket A_1,\ldots,A_m \,|\, S \rrbracket \doteq \llbracket B_0,\ldots,B_n \,|\, S' \rrbracket$), and

(iii): investigating $\llbracket A_0,\ldots,A_m \,|\, S \rrbracket \doteq \llbracket B_1,\ldots,B_n \,|\, S' \rrbracket$,

of action (8). They return $f(m,n)$, $f(m,n+1)$, and $f(m+1,n)$ solutions, respectively.

4.13 □

It is immediate to verify that such number of solutions is exactly the minimum number of solutions needed. However, the algorithm it returns five solutions to $\llbracket A, A \,|\, S \rrbracket \doteq \llbracket A \,|\, S' \rrbracket$, whereas only the three solutions $S \doteq S'$, $S \doteq \llbracket A \,|\, S' \rrbracket$, and $S' \doteq \llbracket A \,|\, S \rrbracket$ are needed.

4.2.4 Hybrid sets unification

To develop a unification algorithm for the well-founded hybrid theory of sets `WF_sets` presented in § 3.2.4, one must combine the ideas used in designing the unification algorithm Unify_bags and Unify_clists presented in § 4.2.2 and § 4.2.3, respectively.

The unification algorithm for hybrid sets is based on the following main program:

```
Unify_set($\mathcal{E}$);
stack := empty;
while $\mathcal{E}$ is not in solved form or not empty(stack) do
  begin
    while not empty(stack) do
```

```
        begin
          $e$ := pop(stack)
          if $e$ is in solved form with respect to $\mathcal{E}$ and stack
          then $\mathcal{E} := e \wedge \mathcal{E}$
          else action
        end
      select arbitrarily from $\mathcal{E}$ an equation $e$ not in solved form; action
    end.
```

which is the same algorithm presented for the multi-set case, and on the procedure action described in Figg. 4.4 and 4.5.

Let us briefly comment action (9) of the algorithm. Its aim is the reduction of set-set equations, in particular, cases (ii) and (iii) take care of duplicates in the left-hand side term and in the right-hand side term, respectively (axiom (E_2)). Case (iv), instead, reflects the permutativity of the set constructor $\{\cdot \mid \cdot\}$ (axiom (E_1)).

As an example, let us consider the system $\{a|X\} \doteq \{b, a|Y\}$. The algorithm applies action $(9.a)$, requiring one of the following systems to be solved

	$\mathcal{E}$	stack
(i)	$a \doteq b$	$[X \doteq \{a\|Y\}]$
(ii)	$a \doteq b$	$[\{a\|X\} \doteq \{a\|Y\}]$
(iii)	$a \doteq b$	$[X \doteq \{b, a\|Y\}]$
(iv)	$\emptyset$	$[\{a\|Y\} \doteq \{a\|N\}, X \doteq \{b\|N\}]$

The first three clearly have no solution, whereas system (iv) can be further transformed by applying again action $(9.a)$ to its first equation, which leads to the following new systems:

	$\mathcal{E}$	stack
(i)	$\emptyset$	$[Y \doteq N, X \doteq \{b\|N\}]$
(ii)	$\emptyset$	$[X \doteq \{b\|N\}, \{a\|Y\} \doteq N]$
(iii)	$\emptyset$	$[X \doteq \{b\|N\}, Y \doteq \{a\|N\}]$
(iv)	$\emptyset$	$[N \doteq \{a\|N'\}, Y \doteq \{a\|N'\}, X \doteq \{b\|N\}]$

```
action
  let $\mathcal{E}'$ be $\mathcal{E} \setminus \{e\}$;
  case $e$ of
 (1)  $X \doteq X$  $\mapsto$  $\mathcal{E} := \mathcal{E}'$
 (2)  $t \doteq X$
      $t$ is not a variable  $\mapsto$  $\mathcal{E} := \mathcal{E}' \wedge X \doteq t$
 (3)  $X \doteq t \wedge \mathcal{E}$
      $X$ does not occur in $t$
      $X$ occurs in $\mathcal{E}$  $\mapsto$  $\mathcal{E} := \mathcal{E}'[X/t] \wedge X \doteq t$;
                                              stack := stack$[X/t]$
 (4)  $X \doteq f(t_0, \dots, t_n)$
      $f \not\equiv \{\cdot \,|\, \cdot\}$ and $X \in FV(t_0, \dots, t_n)$  $\mapsto$  fail
 (5)  $X \doteq \{t_0, \dots, t_n \,|\, t\}$
      $t \not\equiv \{\cdot \,|\, \cdot\}$ and ($X \in FV(t_0, \dots, t_n)$ or
      $t \not\equiv X$ and $X$ occurs in $t$)  $\mapsto$  fail
 (6)  $X \doteq \{t_0, \dots, t_n \,|\, X\}$
      $X$ does not occur in $t_0, \dots, t_n$  $\mapsto$
          $\mathcal{E} := \mathcal{E}'$;
          push($X \doteq \{t_0, \dots, t_n \,|\, N\}$, stack)
 (7)  $f(s_1, \dots, s_m) \doteq g(t_1, \dots, t_n)$
      $f$ different from $g$  $\mapsto$  fail
 (8)  $f(s_1, \dots, s_m) \doteq f(t_1, \dots, t_m)$
      $f \in \Sigma\{\{\cdot \,|\, \cdot\}\}$  $\mapsto$
          $\mathcal{E} := s_1 \doteq t_1 \wedge \ldots \wedge s_m \doteq t_m \wedge \mathcal{E}'$
```

Figure 4.4: Set unification algorithm *à la* Robinson–I

```
(9)   $\left.\begin{array}{r}\{t_0,\dots,t_m \,|\, h\} \doteq \{t'_0,\dots,t'_n \,|\, k\} \\ h,k \text{ ur-elements or variables:}\end{array}\right\} \mapsto$
      if $h, k$ are not the same variable
      (9.a) then choose one of the following actions:
         $(i)$    $\mathcal{E} := t_0 \doteq t'_0 \wedge \mathcal{E}'$;
                  $\mathsf{push}(\{t_1,\dots,t_m \,|\, h\} \doteq \{t'_1,\dots,t'_n \,|\, k\}, \mathsf{stack})$
         $(ii)$   $\mathcal{E} := t_0 \doteq t'_0 \wedge \mathcal{E}'$;
                  $\mathsf{push}(\{t_0,\dots,t_m \,|\, h\} \doteq \{t'_1,\dots,t'_n \,|\, k\}, \mathsf{stack})$
         $(iii)$  $\mathcal{E} := t_0 \doteq t'_0 \wedge \mathcal{E}'$;
                  $\mathsf{push}(\{t_1,\dots,t_m \,|\, h\} \doteq \{t'_0,\dots,t'_n \,|\, k\}, \mathsf{stack})$
         $(iv)$   $\mathcal{E} := \mathcal{E}'; \mathsf{push}(\{t_1,\dots,t_m \,|\, h\} \doteq \{t'_0 \,|\, N\}, \mathsf{stack})$;
                  $\mathsf{push}(\{t_0 \,|\, N\} \doteq \{t'_1,\dots,t'_n \,|\, k\}, \mathsf{stack})$;
      (9.b) else $h, k \in \mathcal{V}$, $h \equiv k \equiv X$
      select arbitrarily $i$ in $\{0,\dots,m\}$;
      choose one of the following actions:
         $(i)$    $\mathcal{E} := t_0 \doteq t'_i \wedge \mathcal{E}'$;
                  $\mathsf{push}(\{t_1,\dots,t_m \,|\, h\} \doteq \{t'_0,\dots,t'_{i-1},t'_{i+1},\dots,t'_n \,|\, k\}, \mathsf{stack})$
         $(ii)$   $\mathcal{E} := t_0 \doteq t'_i \wedge \mathcal{E}'$;
                  $\mathsf{push}(\{t_0,\dots,t_m \,|\, h\} \doteq \{t'_0,\dots,t'_{i-1},t'_{i+1},\dots,t'_n \,|\, k\}, \mathsf{stack})$
         $(iii)$  $\mathcal{E} := t_0 \doteq t'_i \wedge \mathcal{E}'$;
                  $\mathsf{push}(\{t_1,\dots,t_m \,|\, h\} \doteq \{t'_0,\dots,t'_n \,|\, k\}, \mathsf{stack})$
         $(iv)$   $\mathcal{E} := \mathcal{E}'; \mathsf{push}(X \doteq \{t_0 \,|\, N\}, \mathsf{stack})$;
                  $\mathsf{push}(\{t_1,\dots,t_m \,|\, N\} \doteq \{t'_0,\dots,t'_n \,|\, N\}, \mathsf{stack})$
```

Figure 4.5: Set unification algorithm *à la* Robinson–II

By variable substitution, they carry to the following four situations:

1 $Y \doteq N, X \doteq \{b \,|\, N\}$
2 $X \doteq \{b, a \,|\, Y\}, N \doteq \{a \,|\, Y\}$
3 $X \doteq \{b \,|\, N\}, Y \doteq \{a \,|\, N\}$
4 $N \doteq \{a \,|\, N'\}, Y \doteq \{a \,|\, N'\}, X \doteq \{b, a \,|\, N'\}$

The substitutions suggested by such solved form equation systems constitute a complete set of unifiers for the initial syste. Note that this set is not minimal even though sound and complete. For instance, the unifier that can be obtained by the fourth disjunct $[X \doteq \{b, a|N'\}, Y/\{a|N'\}]$ is an instance (modulo (E_1) and (E_2)) of the one obtainable by the second disjunct (over the variables of the initial problem) $[X/\{b, a|Y\}]$. This can be seen by applying to it the substitution $[Y/\{a|N'\}]$.

In general, the set of substitutions computed by our unification algorithm can contain substitutions which are less general and/or equivalent (with respect to the given theory) to other substitutions in the set. The number of these 'redundancies' is in any case finite; in § 4.4 it will be shown a technique to optimize the behavior of Unify_set with respect to the minimality property, namely the capability of reducing redundancies.

Equations of the form $\{t_0, \ldots, t_m \,|\, X\} \doteq \{t'_0, \ldots, t'_n \,|\, X\}$, where the two sides are set terms with the same variable tail element, are handled as a special case by action $(9.b)$. Using action $(9.a)$ also to deal with this kind of equations, it is easy to find a sequence of actions leading to non termination. The problem is close to the one described in § 4.2.2 about unification of bags ended by the same variable. Also in this case a data structure *stack* allows to impose as much determinism as needed to guarantee the termination result.

Remark 4.14 *The unification algorithm presented here is akin to the one sketched by Jayaraman and Plaisted in [57, 55], but it solves a larger number of cases. In particular, the algorithm in [57, 55] intentionally does not take into account the* idempotency *property of sets (i.e. our absorption property). While this restriction enables a simplification of the unification algorithm, it leads, on the other hand, to a loss in* *expressivity and flexibility. Furthermore, the algorithm in [57, 55] does not properly take into account the situation dealt with by action* (9.b) *of our algorithm (set terms with the same variable tail).*

The following theorems state soundness, completeness and termination of $\mathsf{Unify\_set}(\mathcal{E})$ for any given system $\mathcal{E}$ of equations.

Theorem 4.15 (Soundness and Completeness) *Given a system $\mathcal{E}$, suppose $\mathcal{E}_1, \dots, \mathcal{E}_n$ $(n \geq 0)$ are all the systems non-deterministically returned by $\mathsf{Unify\_set}(\mathcal{E})$. Then* `WF_sets` $\vdash \mathcal{E} \leftrightarrow \exists X_1, \dots, X_m \bigvee_{i=1}^{n} \mathcal{E}_i$, *where $X_1, \dots, X_m$ are the variables occurring in $\mathcal{E}_1, \dots, \mathcal{E}_n$ but not in $\mathcal{E}$.*

Proof. Actions (1)–(5) and (7)–(8) of the algorithm trivially yield the desired equivalence, in view of the basic properties of equality and of the freeness assumptions (cf. proof of preceding correctness Theorems 4.4, 4.6, and 4.12).

Justifying action (6) is also straightforward, in view of the permutativity and absorption properties of $\{\cdot \mid \cdot\}$.

Action (9.a) is exactly axiom (E_k^s).

The '$\leftarrow$' direction of action (9.b) is justified again by axiom (E_k^s). For the '$\rightarrow$' direction observe that $\{t_0, \dots, t_m \mid X\} \doteq \{t'_0, \dots, t'_n \mid X\}$ and $t_0 \neq t'_i$ for all $i = 0, \dots, n$ implies that $t_0 \in X$. Moreover, interpreting N as X `less` t_0 or as X itself, then also $\{t_1, \dots, t_m \mid N\} \doteq \{t'_0, \dots, t'_n \mid N\}$ must hold.

4.15 □

Theorem 4.15 shows, in particular, that $\{\mathcal{E}_1, \dots, \mathcal{E}_n\}$ is a complete set of $(E_1)(E_2)$-unifiers for $\mathcal{E}$.

Theorem 4.16 (Termination) *For any Herbrand system $\mathcal{E}$, the procedure*$\mathsf{Unify\_set}(\mathcal{E})$ *always terminates, no matter what sequence of non-deterministic choiches is made.*

Proof. The data structure stack is introduced for sequencing the *unify* actions:

action (6)**:** an action (3), performed on the newly generated equation e' immediately follows, unless e' is already in solved form.

action (9)**:** this action generates two new equations, e_1 and e_2, the second of which typically enables action (9) again. In any case, e_2 is immediately treated. When the sequence of consecutive actions so determined reaches an end, a similar '9-exhaustive' treatment of e_1 is triggered in the case of actions $(9.a)iv$ and $(9.b)iv$.

Intuitively, a branch of the Unify_set execution induces a series of modifications on a forest structure (or simply a DAG) with superimposed links, akin to the one in [92]. Initially, this structure $\mathcal{G}_\mathcal{E}$, labeled over the set of logical symbols, represents the collection of all terms appearing as sides of equations in $\mathcal{E}$;[2] moreover, a link connects two leaves in $\mathcal{G}_\mathcal{E}$ if and only if the two are labeled by the same variable. An execution of action (3) does not add new nodes or edges to $\mathcal{G}_\mathcal{E}$: rather, it creates a link between two nodes, representing X and t respectively. When new set terms are generated by action (5) or (9), new nodes are added to $\mathcal{G}_\mathcal{E}$ to represent them.

A *path* through $\mathcal{G}_\mathcal{E}$ is defined to be a list $n_0, \ldots, n_\ell$ of nodes such that each $\langle n_i, n_{i+1}\rangle$ is either a link or an edge (unlike edges, links can be exploited in both directions). By the very construction of $\mathcal{G}_\mathcal{E}$, all paths issuing from a variable will hit the same non-variable term, if any.

Let us indicate how to measure the *cost* of a path. Links, as well as second edges issuing from `with`-nodes, have cost 0; the cost of the remaining edges is 1; the cost of a path is the sum of the costs of all edges and links constituting it, increased by 1 if the path ends in a constant node.[3] A rough explanation of these conventions is that we intend to combine the notion of height of a term with the (equally widespread) notion of rank of a set.

To conclude this preamble, let us define the *pseudorank* of a node n to be the maximum cost of a path issuing from n. Relevant to our analysis of *unify* is that the cost of a path will never exceed the initial number p of non-variable nodes in $\mathcal{G}_\mathcal{E}$. On the one side, this results from the occurrence checks made in actions (3) and (4): these in fact forestall the formation of cyclic paths. On the other side, it follows from inspection of the *unify* algorithm (cf. actions (5), $(9.a)iv$ and

[2]As usual, there will be an ordered m-tuple of edges issuing from a node labeled f/m and there will be no edge issuing from a variable node.

[3]Paths of cost 0 will establish an equivalence relation between nodes.

$(9.b)i, ii, iv)$ that no addition of edges can disrupt this bound on the path length. In fact, each new edge of cost 1 shares the target node with a pre-existing edge of cost 1: this implies that no path will ever contain both such edges at once.

Let $E^{(0)}, E^{(1)}, E^{(2)}, \ldots$ be the successive values of $\mathcal{E}$ along a branch of the *unify* execution not ending with a failure.

Indicating by prk_i the pseudorank function defined on the nodes of $\mathcal{G}_{\mathcal{E}}$ at the time when $\mathcal{E} = E^{(i)}$, it makes sense to define $prk_{\infty}(n)$ to be the ultimate value of the sequence

$$prk_i(n), prk_{i+1}(n), \ldots\ ,$$

for any node n eventually introduced in $\mathcal{G}_{\mathcal{E}}$. This sequence, being bounded by p as we have already noticed, can in fact increase at most p times.

Plainly, every prk_i (even the one with $i = \infty$) satisfies the identities

$$\begin{array}{rcl} prk_i(f(t_1, \ldots, t_m)) & = & 1 + \max_{j=1}^{m} prk_i(t_j)\ , \\ prk_i(\{t \mid s\}) & = & \max\{1 + prk_i(t), prk_i(s)\}\ . \end{array}$$

We can exploit the 'limit pseudorank' $prk \equiv prk_{\infty}$, whose domain is enlarged by putting $prk(\ell \doteq r) = prk(\ell) + prk(r)$, to associate with $\mathcal{E}$ a useful $(2 \cdot p + 1)$-tuple $Size(\mathcal{E})$ of non-negative integers. Indicating by $\lhd$ the lexicographic ordering, we will prove that $Size(E^{(i)}) \lhd Size(E^{(i+1)})$ unless the action leading from $E^{(i)}$ to $E^{(i+1)}$ is one of the actions (2), (6), or (9). Even then, a decrease will turn out to be the outcome of a 'phase' consisting of consecutive actions. In particular, in the case of actions (6) and (9), the phase is the series of actions imposed by the execution strategy described at the beginning. It will follow, thanks to the well-foundedness of $\lhd$, that the $E^{(i)}$ sequence eventually terminates, which proves our thesis.

It is helpful to view $\mathcal{E}$ as composed of two disjoint collections $\mathcal{E}_1$, $\mathcal{E}_2$ of equations: $\mathcal{E}_2$ is the part of the system which has already been brought to solved form; the remaining equalities form the 'working system' $\mathcal{E}_1$.[4] The definition of *Size* is

$$Size(\mathcal{E}) = \langle\, |\{e \,\mathrm{in}\, \mathcal{E}_1 \;:\; prk(e) = 2\cdot p\}|, \ldots, |\{e \,\mathrm{in}\, \mathcal{E}_1 \;:\; prk(e) = 0\}|\,\rangle\,.$$

We now proceed to show that every successful phase of *unify* reduces the complexity. In the following, each number on the left indicates the first action of a phase.

(1) One equation with pseudorank $2 \cdot prk(X)$ is removed: the size decreases.

(2) This action occasionally decreases the size, but may also leave it unchanged. Anyway, it is unproblematic, because it cannot be performed an indefinite number of consecutive times. It suffices for our purposes to regard a series of such actions as preamble of the subsequent phase.

(3) One equation $X \doteq t$ with pseudorank $prk(X)+prk(t) = 2\cdot prk(X)$ is moved from $\mathcal{E}_1$ to $\mathcal{E}_2$. This lowers the size.

(6) The presence in the new system of the equation

$$X \doteq \{t_0, \ldots, t_n \,|\, N\}$$

forces $prk(N) \leq prk(X)$ to hold, which implies that the size does not increase in consequence of the replacement of $X \doteq \{t_0, \ldots, t_n \,|\, X\}$ by $X \doteq \{t_0, \ldots, t_n \,|\, N\}$ in $\mathcal{E}_1$. The size will decrease thanks to action (3), which is required to be performed immediately.

(8) One equation of pseudorank

$$2 + \max\{prk(t_1), \ldots, prk(t_n)\} + \max\{prk(t'_1), \ldots, prk(t'_n)\}$$

is replaced by equations of lower pseudorank in $\mathcal{E}_1$: lexicographically, the size tuple decreases.

[4]Conceptually, all equations which are in solved form are moved from $\mathcal{E}_1$ to $\mathcal{E}_2$ before each action; then, after e has been chosen, a (possibly empty) collection of newly generated equations replace it in $\mathcal{E}_1$. As for e, sometimes it gets moved to $\mathcal{E}_2$, sometimes it simply gets discarded.

(9) One can view the global effect of the phase starting with action (9), as that of replacing the selected equation

$$e \equiv \{t_0, \ldots, t_n \mid h\} \doteq \{t'_0, \ldots, t'_m \mid k\}$$

by a collection $t_{i_1} \doteq t'_{j_1}, \ldots, t_{i_p} \doteq t'_{j_p}$ of equations relating elements of the two sets, one or two equations regarding h and k being also added, as explained in detail below.

The pseudorank $\bar{\ell} = \max\{1 + prk(t_0), \ldots, 1 + prk(t_n), prk(h)\} + \max\{1+prk(t'_0), \ldots, 1+prk(t'_m), prk(k)\}$ of e, exceeds that of any $t_i \doteq t'_j$ equation. Hence we only need to focus on the equations concerning h and k. Various cases need to be considered.

- If neither h nor k is a variable, there will be only one equation e' regarding h, k. If the form of e' differs from $h \equiv f(r_1, \ldots, r_q) \doteq f(r'_1, \ldots, r'_q) \equiv k$, an immediate failure will ensue due to action (7). Else, since the pseudorank of e' cannot exceed $\bar{\ell}$, action (8), immediately performed on e', will cause the size to decrease.
- If h is a variable and k is not, the only case that does not immediately lead to failure or size decrease, is when the new equation e' has the form $h \doteq \{t'_{i_1}, \ldots, t'_{i_q} \mid k\}$, with h not occurring in the right-hand side, and e' gets added to the working system $\mathcal{E}_1$. Action (3), immediately performed on e', will cause the size to decrease. The case when k is a variable and h is not is entirely analogous.
- If h and k are both variables, with $h \not\equiv k$, the equation(s) dealing with them added to the working system can be of the form:
  - $h = \{t'_{i_1}, \ldots, t'_{i_q} \mid k\}$, or
  - $k = \{t_{i_1}, \ldots, t_{i_q} \mid h\}$, or
  - $h = \{t'_{i_1}, \ldots, t'_{i_q} \mid N\}$, and $k = \{t_{j_1}, \ldots, t_{j_r} \mid N\}$, with N new variable.

  After action (3) is performed (possibly twice), the size decreases.[5]

[5]Note how important is the assumption that h differs from k: without it —in

- If h and k are the same variable X (action (9.*b*) of *unify*), an equation of the form $X \doteq \{t_{i_1}, \ldots, t_{i_q}, t'_{j_1}, \ldots, t'_{j_{q'}} \mid X\}$ (possibly of pseudorank $\bar{\ell}$) is added to the working system. After actions (6) and (3) are performed on this equation, the size decreases.

4.16 □

4.3 The non-well-founded case

As shown at the beginning of § 4.2, all unification algorithms *à la Robinson* share the common unpleasant effect to require an exponential space when the input is of a certain form. This problem has been solved in literature, producing unification algorithms almost linear [77] and even linear [92]. As a starting point, we select the algorithm presented in [78], which simplifies the famous Martelli and Montanari algorithm ([77]). All the algorithms that will be presented can be used to solve also the well-founded problem, provided they are ended by a suitable acycicity test.

Observe that in this section the subsection related to multi sets follows all the other. This has been done to improve readability. In fact, the algorithm for bags requires some concepts developed for the set case.

A pre-processing of the system is required in order to produce an equivalent *flat* system.

Definition 4.17 *An equation set $\mathcal{E}$ is said to be* FLAT *if every equation in it is of the form*

- $X = Y$*, or*
- $X = f(Y_1, \ldots, Y_n)$.

the third case—, after the substitution $\{h \leftarrow \{t'_{i_1}, \ldots, t'_{i_q} \mid N\}\}$ is performed, the second equation would become a set-set equation, possibly of pseudorank $\bar{\ell}$.

Such pre-processing can be performed by the following simple algorithm:

```
while $\mathcal{E}$ is not flat do
  begin
    if $\ell \doteq r \in \mathcal{E}$ and $\ell$ is not a variable
    then replace $\ell = r$ with $N \doteq \ell$ and $N \doteq r$, $N$ a new variable;
    if $X \doteq f(\dots, t, \dots) \in \mathcal{E}$ and $t$ is not a variable
    then replace $X \doteq f(\dots, t, \dots)$ with
        $X \doteq f(\dots, T, \dots)$ and $T \doteq t$, $T$ a new variable
  end.
```

As in the previous section, Capital letters X, Y, Z, etc. represent variables; f, g, etc. stand for functional symbols (i.e. elements of Σ); $\equiv$ denotes the syntactic identity relation between first-order terms over $\Sigma \cup \mathcal{V}$; φ^X_Y denotes the result of replacing every occurrence of the variable X by Y in a quantifier-free first-order expression φ, and $FV(\varphi)$ denotes the set of all free variables occurring in φ.

There are systems of equations of special forms for which a solution can be determined quite easily.

Definition 4.18 *A Herbrand system $\mathcal{E}$ is said to be in* SOLVABLE FORM *if each equation in it has one of the forms:*

- *$X \doteq Y$, with Y distinct from X and X not occurring elsewhere in $\mathcal{E}$;*
- *$X \doteq f(Y_1, \dots, Y_n)$, $f \in \Sigma$, X not occurring elsewhere in $\mathcal{E}$ as l.h.s. of an equation.*

A Herbrand system in solvable form is said to be EXPLICIT *if it contains no subsystem of the following* ZIPPER *form:*

$$X_0 \doteq \{Y_0 \,|\, X_1\}, \dots, X_{m-1} \doteq \{Y_{m-1} \,|\, X_m\}, X_m \doteq \{Y_m \,|\, X_0\}$$

similarly for $[\cdot \,|\, \cdot]$, $\{\!\!\{ \cdot \,|\, \cdot \}\!\!\}$, and $[\![\cdot \,|\, \cdot]\!]$. For $m = 0$ this reduces to the single equation $X_0 \doteq \{Y_0 \,|\, X_0\}$.

For the set case, from any non-explicit solvable form system, it is easy to obtain an equivalent explicit one. This can not be done in the case of lists, bags, and compact lists, since explicit solvable form systems admit finite solutions, and non-explicit only infinite ones.

We are now ready to state the unification problem in very specific terms. Systems in solvable form can, for that sake, be employed as '*templates*' of the solutions to a given system:

Definition 4.19 *Given a Herbrand system $\mathcal{E}$,* SOLVING *$\mathcal{E}$ amounts to producing a finite set of Herbrand systems in solvable form $\mathcal{E}_1, \dots, \mathcal{E}_m$, such that*

- *for every solution γ to $\mathcal{E}$, at least one of the $\mathcal{E}_i$'s has a solution σ such that $\gamma(X) = \sigma(X)$ for all X in $FV(\mathcal{E}) \cap FV(\mathcal{E}_i)$;*
- *for any solution σ of any to the $\mathcal{E}_i$'s, every substitution γ such that $dom(\gamma) \supseteq FV(\mathcal{E})$ and $\gamma(X) = \sigma(X)$ for all X in $FV(\mathcal{E}) \cap \mathrm{dom}(\sigma)$, is a solution to $\mathcal{E}$.*

4.3.1 NWF-list unification

The unification algorithm for non-well-founded, even infinite hybrid lists (or, simply, for infinite terms, as named in [78]) can be stated as follows.

```
nwf_Unify_lists($\mathcal{E}$);
  (1)   $X \doteq X \wedge \mathcal{E} \mapsto \mathcal{E}$
  (2)   $\left.\begin{array}{r} X \doteq Y \wedge \mathcal{E} \\ X \text{ occurs in } \mathcal{E} \end{array}\right\} \mapsto$
            $\mathcal{E}[X/Y] \wedge X \doteq Y$
  (3)   $\left.\begin{array}{r} X \doteq f(Y_1,\dots,Y_n) \wedge X \doteq g(Z_1,\dots,Z_n) \wedge \mathcal{E} \\ f \not\equiv g \end{array}\right\} \mapsto \mathsf{fail}$
  (4)   $X \doteq f(Y_1,\dots,Y_n) \wedge X \doteq f(Z_1,\dots,Z_n) \wedge \mathcal{E} \mapsto$
            $\mathcal{E} \wedge X \doteq f(Z_1,\dots,Z_n) \wedge Y_1 \doteq Z_1 \wedge \dots \wedge Y_n \doteq Z_n$
```

it is important to notice that it can become an efficient unification algorithm for finite terms, simply performing an acyclicity test to the graph related to the (solvable form) system returned as result by the algorithm.

Termination, correctness and completeness are easy to prove:

Theorem 4.20 (Termination) nwf_Unify_lists($\mathcal{E}$) *performs at most*

$$(e+1)(v+s\times\alpha)$$

actions, where v, s, and e are the number of variables, of occurrences of functional symbols, and of equations in $\mathcal{E}$, respectively, and $\alpha = \max\{ar(f) : f \in \Sigma\}$.

Proof. Action (3) causes (failing) termination. It can be executed at most once.

Action (4) removes an occurrence of the functional symbol f from the system $(ar(f) = n)$. It can be executed at most s times.

Action (2) causes the variable X becomes eliminable, namely, it will never occur in an equation firing an action of nwf_Unify_lists. It can be executed at most v times.

Action (1) is fired by (trivial) equations of the form $X \doteq X$. At the beginning there are at most e equations of this form. Any substitution application (action (2)) can introduce at most one of these equations. Any term decomposition (action (4)) at most α, where $\alpha = \max\{ar(f) : f \in \Sigma\}$.

4.20 □

Theorem 4.21 (Complexity) *A simple optimization of the procedure* nwf_Unify_lists *performs at most* $(2\cdot v)+s+e$ *actions to compute* nwf_Unify_lists($\mathcal{E}$).

Proof. We define such optimization as follows:

- execute first action (1) as much as possible (at most e times);
- when action (2) is executed, select (if any) the equation $Y \doteq Y$, just introduced (at most v times) and apply action (1);
- do not introduce in $\mathcal{E}$ equations of the form $X \doteq X$ when generated by action (4).

From the proof of Theorem 4.20, the result follows trivially.

4.21 □

Axiom schema (F_4) (cf. § 3.2) ensures that a solvable form system is satisfiable in the theory `NWF_lists`. Moreover,

Theorem 4.22 (Soundness and Completeness) *Let $\mathcal{E}'$ be the solvable form system produced by* nwf_Unify_lists($\mathcal{E}$)*. Then:*

$$\texttt{NWF\_lists} \vdash \mathcal{E} \leftrightarrow \mathcal{E}' \,.$$

Proof. As usual, we prove this fact by showing correctness and completeness of each action of nwf_Unify_lists. The global result follows from the termination Theorem 4.20.

Action (1) is justified by axiom ($\doteq_1$); action (2) by ($\doteq_2$).

Correctness of action (4) is justified by axiom (F_1); completeness by ($\doteq$).

Action (3) is justified by axiom (F_2).

4.22 □

4.3.2 NWF-compact-list unification

In what follows we show how to extend the simple unification algorithm for `NWF_lists` to the non-well-founded theory of hybrid compact-lists `NWF_clists` presented in § 3.2.3. Similarly to what done in § J4.2.3, we need to modify the term decomposition action (4) with a non-deterministic one able to deal with the semantics of the compact-list constructor symbol $[\![\, \cdot \,|\, \cdot \,]\!]$, regulated by axiom ($E^c_k$).

Axiom (F^c_4) allows to deal with infinite compact-lists (cf. § 3.2.3). However, if one is interested only in finite (non-well-founded) solutions, it is sufficient to include the optional action (3) that rejects infinite solutions. Such action must be included aiming at using nwf_Unify_clists as a unification algorithm for the well-founded theory (in this case it must be ended by a occur check phase). Termination and computational complexity of the algorithm is not affected by the presence of action (3). Assuming it, observe that only simple *zippers* of the form $X \doteq [\![\, Y \,|\, X \,]\!]$ can occur in the final system. However, they admit the solution $[X/[\![\, Y \,|\, t \,]\!]]$, for each term t, even ground and finite.

Given a system $\mathcal{E}$, after it has been reduced to the flat form, the following algorithm reduces it to an equivalent disjunction of flat systems in solvable form. We introduce a little determinism so as to logically split the algorithm of Fig. 4.6 into three parts. In the first part (the

Preamble) and the second one (the Turning point) the order of execution is important. In the third (Action) the ordering is immaterial. Moreover, in this part a *don't know* non-determinism is introduced by action (b).

Before we undertake analyzing the complexity of nwf_Unify_clists —to be followed by the correctness proof—, let us introduce a little nomenclature.

Definition 4.23 *It will be useful to think of a (non-deterministic) execution of* nwf_Unify_clists *as consisting of segments classified as* PHASES*. A* PHASE *is an iteration of the outer repeat, consisting of a preamble (i.e., a full execution of the inner repeat), followed by the checks performed at the turning point, and (unless a termination has occurred) by one of the actions in the* action *part (when more than one of these is viable, one is arbitrarily chosen).*

Theorem 4.24 (Termination) *Let $\mathcal{E}$ be a flat system, and let v and s be the number of distinct variables and the number of occurrences of functional symbols in $\mathcal{E}$, respectively. Then every non deterministic branch of the computation of* nwf_Unify_clists($\mathcal{E}$) *terminates in less than s phases.*

Proof. Any phase consists, in particular, of an execution of (exactly) one of actions (a), (b.1), (b.2), and (b.3). The execution of any of such actions lowers the number of occurrences of functional symbols in $\mathcal{E}$.

To conclude, notice that no action in the Preamble (or Turning point) increases the number of occurrences of functional symbols.

4.24 □

Introducing the simple optimization described inside Theorem 4.21, whose aim is to give a clever handling of trivial equations $X \doteq X$, one immediately has that

Corollary 4.25 *A simple optimization of* nwf_Unify_clists *performs at most $(2 \cdot v + s) + e$ actions to compute* nwf_Unify_clists($\mathcal{E}$). □

This fact, together with the result of above Theorem 4.24, are a proof of the following:

```
nwf_Unify_clists($\mathcal{E}$);
repeat
Preamble: repeat
   (1)   $X \doteq X \wedge \mathcal{E} \mapsto \mathcal{E}$
   (2)   $\left.\begin{array}{r} X \doteq Y \wedge \mathcal{E} \\ X \text{ occurs in } \mathcal{E} \end{array}\right\} \mapsto \mathcal{E}[X/Y] \wedge X \doteq Y$
   (3)$^{(opt)}$   $\left.\begin{array}{r} \text{if there is a zipper } X_0 \doteq [\![ Y_0 \,|\, X_1 ]\!], \ldots, \\ X_{m-1} \doteq [\![ Y_{m-1} \,|\, X_m ]\!], X_m \doteq [\![ Y_m \,|\, X_0 ]\!] \\ m > 0,\ X_0, \ldots, X_m \text{ distinct from one another} \end{array}\right\} \mapsto$
         $\mathcal{E} \wedge X_1 \doteq X_0 \wedge \ldots \wedge X_m \doteq X_0 \wedge$
         $Y_1 \doteq Y_0 \wedge \ldots \wedge Y_m \doteq Y_0$
until nothing has been changed during the last iteration
Turning point:
   Fail:   $\left.\begin{array}{r} X \doteq f(Y_1, \ldots, Y_n) \wedge X \doteq g(Z_1, \ldots, Z_n) \wedge \mathcal{E} \\ f \not\equiv g \end{array}\right\} \mapsto$ fail
   Succeed:  if $\mathcal{E}$ is in solvable form, then exit with success returning $\mathcal{E}$;
Action: perform, non-deterministically, one of the following actions
   $(a)$   $\left.\begin{array}{r} X \doteq f(Y_1, \ldots, Y_n) \wedge X \doteq f(Z_1, \ldots, Z_n) \wedge \mathcal{E} \\ f \in \Sigma \setminus \{[\![ \cdot \,|\, \cdot ]\!]\} \end{array}\right\} \mapsto$
         $\mathcal{E} \wedge X \doteq f(Z_1, \ldots, Z_n) \wedge Y_1 \doteq Z_1 \wedge \ldots Y_n \doteq Z_n$
   $(b)$   $X \doteq [\![ Y \,|\, V ]\!] \wedge X \doteq [\![ Z \,|\, W ]\!] \wedge \mathcal{E} \mapsto$
         $(b.1)$  $Y \doteq Z \wedge V \doteq W \wedge X \doteq [\![ Z \,|\, W ]\!] \wedge \mathcal{E}$
         $(b.2)$  $Y \doteq Z \wedge V \doteq X \wedge X \doteq [\![ Z \,|\, W ]\!] \wedge \mathcal{E}$
         $(b.3)$  $Y \doteq Z \wedge W \doteq X \wedge X \doteq [\![ Y \,|\, V ]\!] \wedge \mathcal{E}$
forever.
```

Figure 4.6: Non well-founded compact-lists unification algorithm

Theorem 4.26 (Complexity) *The unification problem for hybrid compact-lists belongs to the class NP. Hence, thanks to the result of § 4.1.2, stating its NP-hardness even in the simpler well-founded case, it is NP-complete.* □

Correctness and completeness of the algorithm with respect to the theory `WF_clists` is proved in the following

Theorem 4.27 (Soundness and Completeness) *Let $\mathcal{E}$ be an equation system in flat form, and let $\mathcal{E}_1, \dots, \mathcal{E}_h$ be the solvable form equation systems non-deterministically resulting from the computation of* nwf_Unify_clists($\mathcal{E}$). *Then* `NWF_clists` $\vdash \mathcal{E} \leftrightarrow \exists N_1, \dots, N_k \bigvee_{i=1}^{h} \mathcal{E}_i$, *where $N_1, \dots, N_k$ are the variable occurring in $\mathcal{E}_1, \dots, \mathcal{E}_h$ but not in $\mathcal{E}$.*

Proof. As usual, the proof is performed by case analysis. Actions (1), (2), Fail, and (a) are actions (1)–(4) of algorithm nwf_Unify_lists, whose correctness ha been proved in Theorem 4.22.

Action (b) is exactly axiom (E_k^c).

Correctness of action (3) is trivial. Its completeness follows from the fact that accepting it, we explicitly avoided infinite compact lists.

4.27 □

4.3.3 NWF-set unification

We begin with a brief discussion of how a solution to a system in solvable form can be found when the system is in *explicit* solved form. One can proceed to enlarge the system with all equations $Y \doteq \emptyset$, where each Y is a variable that, although occurring in the system, does not occur as left-hand side of any of its equations. The ground image $\gamma(X)$ of each variable X in a solution γ, can thus be read directly off the system.

More generally, a system in solvable form can be modified until it becomes explicit. Every modification step will introduce new variables; however each solution to the modified system can be restricted to the old variables giving a solution to the preceding system. Thus, at the end, any solution to the explicit system will also be a solution to the original system. To see this, note that

$$\gamma(X_0) = \gamma(X_1) = \cdots = \gamma(X_m)$$

must hold in any solution γ, when there is a zipper

$$X_0 \doteq \{Y_0 \,|\, X_1\}, \ldots, X_{m-1} \doteq \{Y_{m-1} \,|\, X_m\}, \;\; X_m \doteq \{Y_m \,|\, X_0\}.$$

Therefore, as long as a there is a subsystem of this kind, we can replace it by the equations

$$X_0 \doteq \{Y_0 \,|\, K_0\}, \ldots, X_m \doteq \{Y_m \,|\, K_m\} \,,$$

where $K_0, \ldots, K_m$ are new variables. While the system resulting from this modification is still in solvable form, it is closer to explicit form, because the number of zippers has been reduced.

Special chains of inclusions, introduced by the following definition, will play an important role in our subsequent discussion:

Definition 4.28 *A* PATH *in a Herbrand system $\mathcal{E}$ is a sequence $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_n}{\leftarrow} X_{n+1}$ of equations such that $X_i \doteq \{Y_i \,|\, X_{i+1}\}$ is in $\mathcal{E}$ for all i* in $\{0, \ldots, n\}$.

We are now beginning to discuss the unification algorithm nwf_Unify-_sets described in Fig. 4.7. Al the other algorithms for the non-well-founded case, it gets an input $\mathcal{E}$ which, without loss of generality, is assumed to be flat.

nwf_Unify_sets performs a non-deterministic search. Reaching the leaf of a successful branch of the search tree, it will output a system $\mathcal{E}_i$ in solvable, explicit, form. The whole search tree will be finite.

The algorithm makes also use of an auxiliary data structure, $\mathcal{C}$, to keep track of a number of temporary assumptions of the form $W \notin V$. Action Fail_1 may detect a failure situation by checking whether a constraint $Y \notin X$ is in $\mathcal{C}$, and the variable Y is forced to belong to X by $\mathcal{E}$. It can be use as an efficient unification algorithm for the theory `WF_sets` simply ending it with a procedure which rejects cyclic solutions.

```
nwf_Unify_sets($\mathcal{E}$: Herbrand_system);
$\mathcal{C} := \emptyset$;
repeat
Preamble: repeat
    (1)   $X \doteq X \wedge \mathcal{E} \;\mapsto\; \mathcal{E}$
    (2)   $X \doteq Y \wedge \mathcal{E}$                       $\mathcal{E}[X/Y] \wedge X \doteq Y$
          $X$ occurs in $\mathcal{E}$   $\}\;\mapsto$            $\mathcal{C} := \mathcal{C}[X/Y]$
                                      if there is a zipper
    (3)   $X_0 \doteq \{Y_0 \,|\, X_1\}, \cdots, X_m \doteq \{Y_m \,|\, X_0\} \wedge \mathcal{E}$   $\}\;\mapsto$
              $X_0, \ldots, X_m$ distinct from one another
                $\mathcal{E} \wedge X_1 \doteq X_0 \wedge \ldots \wedge X_m \doteq X_0 \wedge$
                     $X_0 \doteq \{Y_0 \,|\, K_0\}, \cdots, X_m \doteq \{Y_m \,|\, K_m\}$
                $\mathcal{C} := \mathcal{C} \cup \{Y_0 \notin K_0, \ldots, Y_m \notin K_m\}$
            if there is a path in $\mathcal{E} \wedge X_0 \doteq \{Y_0 \,|\, X_1\}$
    (4)         $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_n}{\leftarrow} X_{n+1} \overset{Y_0}{\leftarrow} X_{n+2}$   $\}\;\mapsto$
          with $Y_0, \ldots, Y_n$ distinct from one another
          $\mathcal{E} \wedge \{X_1 \doteq X_0\}$
    (5)   close $\mathcal{C}$ with respect to the rule:
                $(Y \notin X \; in \, \mathcal{C}, X \overset{Y}{\leftarrow} V \; in \, \mathcal{E}) \Rightarrow Y \notin V \; in \, \mathcal{C}$
until nothing has been modified by the last iteration;
Turning point:
  Fail_1: if there is an edge $X \overset{Y}{\leftarrow} V$ in $\mathcal{E}$ with $Y \notin X \; in \, \mathcal{C}$,
      then exit with failure;
  Fail_2: if there are equations $X \doteq f(X_1, \ldots, X_n)$ and
      $X \doteq g(Y_1, \ldots, Y_m)$ in $\mathcal{E}$ with $f \not\equiv g$, then exit with failure;
  Succeed: if $\mathcal{E}$ is in solvable form, then exit with success returning $\mathcal{E}$;
  Actions:
  $(a)$   $X \doteq f(X_1, \ldots, X_n) \wedge X \doteq f(Y_1, \ldots, Y_n) \wedge \mathcal{E} \;\mapsto$
        $X \doteq f(Y_1, \ldots, Y_n) \wedge \mathcal{E} \wedge X_1 \doteq Y_1 \wedge \ldots \wedge X_n \doteq Y_n$
  $(b)$   $X \doteq \{Y \,|\, V\} \wedge X \doteq \{Z \,|\, W\} \wedge \mathcal{E}$
        ($X \not\equiv V$ and $X \not\equiv W$–ensured by action (3)   $\}\;\mapsto$
          $(b.1)$  $Y \doteq Z \wedge V \doteq W \wedge X \doteq \{Z \,|\, W\} \wedge \mathcal{E}$
          $(b.2)$  $X \doteq V \wedge X \doteq \{Y \,|\, V\} \wedge X \doteq \{Z \,|\, W\} \wedge \mathcal{E}$
          $(b.3)$  $X \doteq W \wedge X \doteq \{Y \,|\, V\} \wedge X \doteq \{Z \,|\, W\} \wedge \mathcal{E}$
          $(b.4)$  $V \doteq \{Z \,|\, N\} \wedge W \doteq \{Y \,|\, N\} \wedge X \doteq \{Z \,|\, W\} \wedge \mathcal{E}$
                   $\mathcal{C} := \mathcal{C} \cup \{Y \notin N, Z \notin N\}$
forever.
```

Figure 4.7: Hyperset unification algorithm

Before we undertake analyzing the complexity of nwf_Unify_sets—to be followed by the correctness proof—, let us develop a little nomenclature and some preliminary remarks and comments.

We extend here the partition of a (non-deterministic) execution of nwf_Unify_sets into segments given in Def. 4.23 with the notion of *stage*.

Definition 4.29 *A* Stage *is a series of consecutive phases that*

- *immediately follows either initialization or an execution of action* (b.4)*;*
- *does not comprise any execution of action* (b.4)*;*
- *either goes on forever, or terminates at the turning point, or ends with an execution of action* (b.4)*.*

(An important fact we will discover later on is that no stage consists of infinitely many phases).

As we are about to discuss, we can think that an equivalence relation between variables is being implicitly calculated by nwf_Unify_sets. Initially, each variable makes an equivalence class by itself; then, the equivalence relation gets refined by the preamble of each phase. Once they have become equivalent, two variables are to represent the same hyperset; accordingly, as soon as a variable formerly generated by action 3 or (b.4) becomes equivalent to an initial variable, we identify the two with one another and cease to regard the generated variable as 'new' any more.

Let us now clarify the main purpose of each preamble. Such purpose is to decompose the system $\mathcal{E}$ into subsystems of the form

$$\left.\begin{array}{c} X_1 \\ \vdots \\ X_\ell \end{array}\right\} \doteq R \doteq \left\{\begin{array}{cc} f_1(Y_{11}, \dots, & Y_{1a_1}) \\ \vdots & \vdots \\ f_m(Y_{m1}, \dots, & Y_{ma_m}) \end{array}\right.$$

with $\ell + m \geq 1$ (hopefully with $f_i \equiv f_j$ for all pair i, j). These subsystems will be mutually independent in the sense that

- no left-hand side variable X_i appears, globally, more than once;

- no *representative* variable R occurs as left-hand side of a variable-variable equation; apart from this, R is entirely free to occur in right-hand sides.

Notice that if $m \leq 1$ held for each one of the said subsystems, then a solvable form would result from the preamble; otherwise an action aimed at reducing some ℓ will immediately take place.

As for the equivalence relation over variables hinted at above, it can be characterized at the end of each preamble as being the reflexive and transitive closure of the relation $\{[X, R] : X \doteq R \text{ in } \mathcal{E}\}$.

In sight of proving that every branch of the search tree of nwf_Unify-_sets eventually breaks off, reporting either a failure or a success, it is useful to associate with the input system $\mathcal{E}$ two size parameters v_0 and s_0:

- v_0 is the number of initial variables;
- s_0 is the number of occurrences of functional symbols in $\mathcal{E}$ (including $\{\cdot\,|\,\cdot\}$).

The following remark will turn out to be the key in proving termination (and complexity) of nwf_Unify_sets.

Remark 4.30 *1. A new variable Q always shows up at creation time in a context $V \doteq \{Y\,|\,Q\}$ (where $V \not\equiv Q$ and $Y \not\equiv Q$); later on, it can be moved to a different context by actions* (2)*, or* $(b.2)$*–*$(b.3)$*. However, it will never come to occupy a label position over an edge $X \overset{Q}{\leftarrow} W$: that is, no equality $X \doteq \{Q\,|\,W\}$ will ever appear into $\mathcal{E}$ unless after Q has become equivalent to a pre-existing label, in which case we convene to no longer regard it as 'new'.*

2. It follows from the previous observation and from the presence of actions (3) *and* (4) *in* nwf_Unify_sets *that no path $X_0 \overset{Y_1}{\leftarrow} X_1 \overset{Y_2}{\leftarrow} \cdots \overset{Y_n}{\leftarrow} X_n$ can have length $n > v_0$ at the end of any preamble.*

3. If an edge $X_0 \overset{Y_0}{\leftarrow} X_1$ drawing its origin from an action (3) *or* $(b.4)$ *becomes part of a path $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_0}{\leftarrow} X_{n+1}$ which* *either has $X_0 \equiv X_{n+1}$ (zipper case) or can be lengthened with an edge $X_{n+1} \overset{Y_{n+1}}{\longleftarrow} X_{n+2}$ already available in $\mathcal{E}$, a failure will take place at the next turning point. This follows from the fact that the creation of the edge $X_0 \overset{Y_0}{\longleftarrow} X_1$ causes the constraint $Y_0 \notin X_1$ to be put in $\mathcal{C}$; however the path in question will cause action* (3) *or* (4) *to trigger a* Fail_1 *termination when the next turning point is reached.*

4. *The preceding observation implies that the overall number of arcs introduced by action* (3) *is less than the number of occurrences of* $\{\cdot\,|\,\cdot\}$ *in the initial system, bounded by* s_0*.*

In what follows we first show that the algorithm terminates provided action $(b.4)$ is performed a finite number k of times (cf. Corollary 4.32). Later on, with Lemma 4.33 we will place a finite bound on k, and with Theorem 4.42 we will refine it into a polynomial bound.

Lemma 4.31 *Consider a stage of a computation of* nwf_Unify_sets($\mathcal{E}$)*. Let v and s be the number of inequivalent variables (initial or not) and the number of occurrences of functional symbols in $\mathcal{E}$, as they are at the beginning of that stage. Then the stage comprises at most $1 + v + s_0 + s$ phases.*

Proof. First note that no action other than $(b.4)$ increases s.

The number v might increase due to an action (3), which however cannot be exploited more than s_0 times, as noticed in Remark 4.30.4. This potential increase of v accounts for the addendum s_0 in the thesis of the lemma.

We can henceforth focus on actions (a), $(b.1)$–$(b.3)$, namely the ones from which a non-deterministic choice is performed in every non-final phase.

(a) and $(b.1)$ cause s to decrease; actions $(b.2)$ and $(b.3)$, although leaving s unchanged, set the ground for either a reduction of v in the next preamble or a Fail_1 termination at the next turning point.

4.31 □

Corollary 4.32 *Suppose action* $(b.4)$ *is performed exactly* k *times during a computation of* nwf_Unify_sets$(\mathcal{E})$. *Then the computation comprises at most* $(k+1)\cdot v_0 + (k+2)\cdot s_0 + (k+1)^2$ *phases.*

Proof. Let $N_1, \ldots, N_k$ be the variables introduced by the successive executions of action $(b.4)$. We can split the sequence of phases performed by nwf_Unify_sets$(\mathcal{E})$ into $k+1$ successive stages determined by the introduction of the N_is.

$$\begin{tikzcd}
\bullet \arrow[r] & \overset{N_1}{\bullet} \arrow[r] & \cdots \arrow[r] & \overset{N_k}{\bullet} \arrow[r] & \circ
\end{tikzcd}$$

The first stage is based on a system with size parameters v_0 and s_0; hence, by Lemma 4.31, it contains no more than $1 + v_0 + s_0 + s_0$ phases. After the application of action $(b.4)$, both s and v get increased at most by one; therefore the second stage, again by Lemma 4.31, contains no more than $1 + (v_0+1) + (s_0+1) + s_0$ phases. JThe overall situation is summarized by the following diagram:

$$\begin{tikzcd}
\bullet \arrow[r, "\underbrace{\hphantom{xxxx}}_{1+v_0+s_0+s_0}"'] & \overset{N_1}{\bullet} \arrow[r] & \cdots \arrow[r] & \overset{N_k}{\bullet} \arrow[r, "\underbrace{\hphantom{xxxx}}_{1+(v_0+k)+(s_0+k)+s_0}"'] & \circ
\end{tikzcd}$$

The number of phases forming a whole computation is hence bounded by $\sum_{i=0}^{k}(1 + (v_0+i) + (s_0+i) + s_0)$; however, it is appropriate to consider the addendum s_0 only once, because it represents an upper bound on the number of edges introduced by action (3), which is a global quantity. Thus, no more than $(k+1)\cdot v_0 + (k+2)\cdot s_0 + (k+1)^2$ phases can be performed.

4.32 □

To place a bound on the number k of times $(b.4)$ gets executed, we will define in terms of $\sim_i$ a tuple τ_i of natural numbers so that modifications of $\mathcal{E}, \mathcal{C}$ made during the i-th phase may cause τ_{i+1} to differ from τ_i. Changes of τ will invariably lower it with respect to the lexicographic well-ordering of tuples, without affecting its lenght. τ gets lowered whenever a new variable is introduced, which happens, in particular, with $(b.4)$. This by itself ensures—in view of Corollary 4.32—the termination of nwf_Unify_sets.

Notice that any lowering of a tuple $[x_0, \ldots, x_\ell]$ with respect to the said ordering can be achieved by elementary *decrease actions* of the following kind:

- a component x_j, with $x_j > 0$, gets decremented by one; simultaneously (if $j < \ell$)
- a component x_i, with $i > j$, is incremented by h units, with $h \geq 0$.

In the case at study, we will be able to show that such decrease actions will always have $0 \leq h \leq 2$; correspondingly, a decrease action will be named:

destruction: if $h = 0$,

climbing: if $h = 1$,

generation: if $h = 2$.

To introduce τ, let us consider the set

$$L_i = \big\{ Y \;:\; X = \{ Y \,|\, W \} \;\text{ in }\; E_i \}$$

of all Ys occurring in a context $X \overset{Y}{\leftarrow} W$ within E_i. By Remark 4.30.1–2, $|L_i| \leq |L_0|$ for all $i \geq 0$. Moreover, as easily seen, $|L_i| = |L_0/\sim_i|$. We define τ_i to be the tuple $\tau_i = [x_0, \ldots, x_{|L_0|}]$, where each x_j is the cardinality of the following set:

$$\left\{ \begin{array}{l} \text{equations of the form } \_ = \{ \_ \,|\, W \} \text{ in } E_i\text{:} \\ |L_0| - |L_i| + |\{Y \;:\; Y \notin W \;\text{ in }\; C_i \}| = j \end{array} \right\}$$

The initial value τ_0 of τ clearly has $x_0 \leq s_0$ and $x_1 = \cdots = x_{|L_0|} = 0$.

Lemma 4.33 *A stage always lowers the value of τ except, possibly, in the imminence of a* Fail_1 *exit.*

Proof. Actions (1), (a), $(b.2)$, and $(b.3)$ do not affect τ. Action $(b.1)$ always affects τ as a destruction action.

The complexity function $compl(\tau)$ has been carefully chosen in order that action (2), whose effect is a sequence of destruction and generation actions, cannot increase it.

Actions (3) and (5) may leave τ unchanged or affect it as climbing actions; action (4) might cause τ to increase, but in this case Fail_1 will immediately cause termination.

If neither $Y \notin W$ nor $Z \notin V$ belongs to $\mathcal{C}$, then action $(b.4)$ makes τ lower (it can be viewed as a generation action). Unless this is the case, an immediate combination of action (3) with Fail_1 will lead to termination.

4.33 □

Theorem 4.34 (Termination) *Let $\mathcal{E}$ be a flat system. Then the procedure* nwf_Unify_sets($\mathcal{E}$) *always terminates, no matter what sequence of non-deterministic choices has been made.*

Proof. Lemma 4.33 ensures that a only finite number of actions $(b.4)$ occur along a branch of the computation. The claim follows, by Corollary 4.32.

4.34 □

One may wonder whether the above translation of nwf_Unify_sets actions into chains of destruction, climbing, and generation actions, may disclose a time complexity assessment sharper than a simple termination result. Unfortunately, this is not the case.

It can be shown that, starting with a tuple $[x_0, \ldots, x_\ell]$ of natural numbers, no more than $3^\ell \cdot x_0 + 3^{\ell-1} \cdot x_1 + \cdots + 3^0 \cdot x_\ell \leq (x_0 + \cdots + x_\ell) \cdot 3^\ell$ consecutive destruction, climbing, and generation actions can be performed.

This exponential bound cannot be improved significantly without refining the technique: if, as in our case, the initial tuple has the form $[s_0, \underbrace{0, \ldots, 0}_{\ell-1}]$, a chain consisting of $s_0 \cdot (2^\ell - 1)$ generation actions followed by $s_0 \cdot 2^\ell$ destructions is *a priori* conceivable. However, as we will now see, a similar chain does not reflect the behaviour of any concrete nwf_Unify_sets computation.

We will prove now that nwf_Unify_sets belongs to the complexity class NP. It hence follows that the hyperset unifiability problem is NP-complete, in view of the reduction of 3-SAT to set matching shown in § 4.1.3.

We will represent $\mathcal{E}$ by a multi-graph $\mathcal{G}_{\mathcal{E}}$, and $\mathcal{C}$ by a function $lev : FV(\mathcal{E}) \longrightarrow \omega$. We momentarily defer the characterization of lev. As for $\mathcal{G}_{\mathcal{E}}$, its constituents are:

- nodes $\mathcal{N}_{\mathcal{E}} = \Big\{ V_1, V_2 \ : \ V_1 = \{\,\_\,|\,V_2\,\} \text{ in } \mathcal{E} \Big\}$;
- labels $\mathcal{L}_{\mathcal{E}} = \Big\{ Y \ : \ \_ = \{\,Y\,|\,\_\} \text{ in } \mathcal{E} \Big\}$;
- directed labelled edges $\mathcal{A}_{\mathcal{E}} = \Big\{ V_1 \overset{Y}{\leftarrow} V_2 \ : \ V_1 = \{Y|V_2\} \text{ in } \mathcal{E} \Big\}$.

Conceptually, the pair $\mathcal{G}_{\mathcal{E}}$, lev gets updated at every turning point, like τ. As we have already proved termination, we are in the position that we can refer to the final value of either of these structures. (Obviously, $\mathcal{G}_{\mathcal{E}}$ and lev, as well as their final values, depend on the course of a non-deterministic computation.)

The following two examples are aimed at conveying an intuitive grasp of why each non-deterministic branch contains a number of phases (equivalently, a number of $(b.4)$ actions) polynomially related to the size $v_0 + s_0$ of $\mathcal{E}$.

Example 4.35 *Starting with the set* $\mathcal{E} = \big\{ X = \{A|V_1\}, X = \{B|V_2\}, X = \{C|V_3\} \big\}$ *of membership constraints, a branch that maximizes the number of stages will yield the following final value for* $\mathcal{G}_{\mathcal{E}}$*. Indicating by the arrow* $\overset{\circ}{\leftarrow}$ *any edge removed by action* $(b.4)$*, we have*

$$\begin{tikzcd}
 & V_1 \arrow[dl, "A"', "\circ" marking] & & \\
X & V_2 \arrow[l, "B"', "\circ" marking] & N_1 \arrow[l, "A"', "\circ" marking] \arrow[ul, "B"'] & \\
 & V_3 \arrow[ul, "C"] & N_2 \arrow[l, "B"'] \arrow[ul, "C"] & N_3 \arrow[l, "A"'] \arrow[ul, "C"]
\end{tikzcd}$$

Notice that the unification algorithm does not generate a variable corresponding to each subset of a three-element set, a reason being that once the edge $X \overset{A}{\leftarrow} V_1$ *has been removed, it becomes impossible to re-exploit it in conjunction with* $X \overset{C}{\leftarrow} V_3$ *to fire an action* $(b.4)$*.*

Example 4.36 *A branch that maximizes the number of phases for the Herbrand system*

$$\begin{array}{rcl}\mathcal{E} & = & \{\ \ X = \{\,Y_1 \mid A_1\,\},\ A_1 = \{\,Y_2\ \mid A_2\,\},\ A_2 = \{\,Y_3 \mid A_3\,\}, \\ & & \quad\ X = \{\,Z_1 \mid B_1\,\},\ B_1 = \{\,Z_2\ \mid B_2\,\}\ \ \},\end{array}$$

expressing the set unification problem $\{Y_1, Y_2, Y_3 \mid A_3\} = \{Z_1, Z_2 \mid B_2\}$, *will yield the following final value for* $\mathcal{G}_{\mathcal{E}}$*:*

$$\begin{tikzcd}
 & & & A_3 \arrow[dl, "Y_3"', "/" marking] & & \\
 & & A_2 \arrow[dl, "Y_2"', "/" marking] & & N_2^{(3)} \arrow[ul, "Z_1"'] \arrow[dl, "Y_3"', "/" marking] & \\
 & A_1 \arrow[dl, "Y_1"', "/" marking] & & N_2^{(2)} \arrow[ul, "Z_1"'] \arrow[dl, "Y_2"', "/" marking] & & N_1^{(4)} \arrow[ul, "Z_2"'] \arrow[dl, "Y_3"'] \\
X & & N_1^{(1)} \arrow[ul, "Z_1"'] \arrow[dl, "Y_1"', "/" marking] & & N_1^{(3)} \arrow[ul, "Z_2"'] \arrow[dl, "Y_2"'] & \\
 & B_1 \arrow[ul, "Z_1"'] & & N_1^{(2)} \arrow[ul, "Z_2"'] \arrow[dl, "Y_1"'] & & \\
 & & B_2 \arrow[ul, "Z_2"'] & & &
\end{tikzcd}$$

Inspection of these examples leads to the following observations:

Remark 4.37 1. *If one subtracts the initial number of edges in* $\mathcal{G}_{\mathcal{E}}$ *from the number of edges in the final* $\mathcal{G}_{\mathcal{E}}$*—counting also those that were removed by* $(b.4)$*—, one obtains twice the overall number* $\overline{\beta}$ *of* $(b.4)$ *actions.*

2. *The evolution of* $\mathcal{G}_{\mathcal{E}}$ *progressively limits the possibility to apply action* $(b.4)$*: calling into play again the decrease actions on the tuple* τ*, this means that generation actions (the only potential source of exponentiality) become less and less viable.*

   Sometimes the termination guarantee does not come from this phenomenon, but, rather, from the presence of constraints. This does not emerge from the two examples just seen, but can be seen from studying a system like the following:

$$X = \{\,\_ \mid A\,\}, X = \{\,\_ \mid B\,\}, Y = \{\,\_ \mid A\,\}, Y = \{\,\_ \mid B\,\}\,.$$

3. *If a node X eventually inserted into $\mathcal{G}_{\mathcal{E}}$ has incoming edges in some phase, at least one incoming edge will always be alive, till the end of the computation.*

It is now time to introduce *lev*:

Definition 4.38 *Given the pair $\mathcal{E}$,$\mathcal{C}$, a* LEVEL-MAPPING *is a function lev : $FV(\mathcal{E}) \longrightarrow \omega$, satisfying the following conditions:*

- *for all edge $V_1 \overset{Y}{\leftarrow} V_2$ of $\mathcal{G}_{\mathcal{E}}$, if $Y \notin V_2$ belongs to $\mathcal{C}$, then $lev(V_2) = lev(V_1) + 1$, otherwise $lev(V_2) \leq lev(V_1) + 1$;*
- *for all equation $V_1 = V_2$ in $\mathcal{E}$, $lev(V_1) = lev(V_2)$.*

Remark 4.39 *In the absence of negative information, a trivial level-mapping can be obtained by simply setting to 0 all variables. On the other hand, it may be the case that a level-mapping does not exist, as the following example shows:*

$$\begin{tikzcd} X & & V \arrow[ll, "Y_1"'] \arrow[dl, "Y_3"'] \\ & W \arrow[ul, "Y_2"] & \end{tikzcd}$$

$Y_3 \notin V$, $Y_2 \notin W$ *both belonging to* $\mathcal{C}$.

Plainly, if a level-mapping lev for $\mathcal{E}$,$\mathcal{C}$ exists at all, it is not unique: one may 'tune' its construction so as it meet the condition

for all variable V, $lev(V) \leq \mid L_{\mathcal{E}} \mid$

This new condition—to be taken from now on as part of the definition of a level-mapping—can easily be fulfilled, e.g., by setting $lev(W) = 0$ for at least one node W in each connected component of (the undirected graph underlying) $\mathcal{G}_{\mathcal{E}}$.

With respect to a level-mapping *lev*, we say that an edge $X \overset{Y}{\leftarrow} V$ (or, more generally, an ordered pair X, V) is of level i if $lev(V) = i$.

Definition 4.40 *Let $\mathcal{E}'$ be the system in solvable form resulting from a (non-deterministic) computation of* nwf_Unify_sets *with input $\mathcal{E}$: hence any variable generated in such a computation occurs in $\mathcal{E}'$. Let, moreover, lev be a level-mapping for $\mathcal{E}'$ (hence for $\mathcal{E}$). Then,*

- *$\alpha(i)$ stands for the number of edges of level i in $\mathcal{G}_{\mathcal{E}}$. Also, $\overline{\alpha} = \sum_{i=0}^{\infty} \alpha(i) = \sum_{i=0}^{|L_{\mathcal{E}}|} \alpha(i)$.*
- *$\beta(i)$ stands for the number of edges of level i introduced in the computation by action $(b.4)$ (disregard of whether such edges survive in $\mathcal{G}_{\mathcal{E}'}$). Let also $\overline{\beta} = \sum_{i=0}^{\infty} \beta(i) = \sum_{i=0}^{|L_{\mathcal{E}}|} \beta(i)$.*

In view of Corollary 4.32, in order to guarantee the NP-completeness of nwf_Unify_sets, it is sufficient to place a polynomial bound on the total number $\frac{\overline{\beta}}{2}$ of executions of $(b.4)$.

Theorem 4.41 *In a successful branch of* nwf_Unify_sets$(\mathcal{E})$*, if no variable X generated by action $(b.4)$ ever joins another variable Y (in the sense that either $X = Y$ or $Y = X$ is put into $\mathcal{E}$), then $\overline{\beta}$ is $O(\overline{\alpha}^3)$.*

Proof. First of all, let us focus on a couple of conditions necessary for an action $(b.4)$ to take place:

- two 'parent' edges of level j concur to the action, where $j + 1$ is the level of the two edges to be created—one of these parent edges will be deleted by the action;
- since the parent edges must enter the same node, the latter cannot have a single incoming edge when the action is fired.

Based on these remarks, we derive the following:

- $\beta(0) = 0$ and $\beta(1) \leq 2 \cdot \alpha(0)$.
- The presence of two generated edges at level $j - 1$ implies that there is a node at level $j - 1$.

  The overall number of generated nodes of level $j - 1$ is $\frac{\beta(j-2)}{2}$; hence the assumption that generated nodes never join—implying that nodes, once generated, persist in $\mathcal{G}_{\mathcal{E}}$ retaining an incoming edge— ensures, for $j \geq 2$, that

$$\beta(j) \leq 2 \cdot \left(\alpha(j-1) + \beta(j-1) - \frac{\beta(j-2)}{2} \right).$$

By unfolding these constraints on β, we obtain

$$\beta(j) \leq 2 \cdot \left(\sum_{i=1}^{j} i \cdot \alpha(j-i) \right) .$$

Since $\mid L_{\mathcal{E}} \mid \leq \alpha$, we conclude:

$$\overline{\beta} = \sum_{j=0}^{|L_{\mathcal{E}}|} \beta(j) \leq 2 \cdot \left(\sum_{j=1}^{|L_{\mathcal{E}}|} \sum_{i=1}^{j} i \cdot \alpha(j-i) \right) = O(\overline{\alpha}^3) .$$

4.41 □

We are left with the task of showing that the assumption that generated variables never join can be discarded from the statement of Theorem 4.41. To this end, notice that two variables of level $j+1$ becoming equivalent might generate a situation in which action $(b.4)$ can be performed. However, an inspection of nwf_Unify_sets (cf. proof of Theorem 4.42 below) shows that at the same time two edges of level j come to coincide. As a consequence, the overall number of actions $(b.4)$ (and hence $\overline{\beta}$) cannot increase.

Theorem 4.42 *In any computation of* nwf_Unify_sets*, no more than* $O(\bar{\alpha}^3) = O\left((s_0)^3\right)$ *new variables can be generated.*

Proof. In view of Theorem 4.41, we only need to show the following fact: if a variable N introduced by action $(b.4)$ as the tail of two edges $V \overset{Y}{\leftarrow} N$ and $W \overset{Z}{\leftarrow} N$ of level i joins another variable A, then a pair of edges of level i has not been used to generate two edges of level $i+1$.

In fact, in order for N to be unified with A, a series $N = A_1, A_1 = A_2, \ldots, A_n = A$ of equalities must be inferred by nwf_Unify_sets. To get the first equation $N = A_1$, the unification algorithm must perform action (b) with one of the two equations $V = \{\, Y \mid N \,\}$ or $W = \{\, Z \mid N \,\}$, together with an equation $V = \{\, C \mid B \,\}$ or $W = \{\, C \mid B \,\}$. A simple inspection of the four subcases of action (b) shows that $(b.1)$ is the only subcase that introduces an equation $N = B$ without leading to a subsequent Fail_1 (note that $B \equiv A$). This means that in such branch of a computation, the edges $V \overset{C}{\leftarrow} B$ (or $W \overset{C}{\leftarrow} B$) and $V \overset{Y}{\leftarrow} N$ (or

$W \overset{Z}{\leftarrow} N$) of level i cannot be used to fire action (b.4) and hence to generate two edges of level $i+1$.

4.42 □

Corollary 4.43 (NP-completeness) *Unification of hybrid hyperset is NP-complete. In particular, every single branch generated during the execution of* nwf_Unify_sets($\mathcal{E}$) *consists in a number of phases polynomially bounded by* $v_0 + s_0$.

Proof. From NP-hardness of set unification problem (hence of hypersets) has been proved in § 4.1. To prove the completeness, notice that Theorem 4.42 implies that the number k of variables generated by action (b.4) of the algorithm is bounded by a polynomial with respect to $v_0 \cdot s_0$. The thesis follows from Corollary 4.32.

4.43 □

It remains to prove that the set unification algorithm nwf_Unify_sets presented above is sound and complete with respect to the axiomatic set theory `NWF_sets`. These important properties of the algorithm can be phrased as follows:

Theorem 4.44 (Soundness and Completeness) *Let* $\mathcal{E}_1, \dots, \mathcal{E}_k$ *be the systems in solvable form produced as output by the algorithm* nwf_Unify_sets($\mathcal{E}$). *Then the following holds*

$$\texttt{NWF\_sets} \vdash \mathcal{E} \leftrightarrow \exists Q_1 \cdots \exists Q_m \bigvee_{i=1}^{k} \mathcal{E}_i \, ,$$

where $Q_1, \dots, Q_m$ *are all the variables in the* $\mathcal{E}_i$*'s not occurring in* $\mathcal{E}$.

In order to achieve greater generality, we will prove the above result as a consequence of an analogous result concerning a variant procedure named nwf_Unify_sets$_*$($\mathcal{E}$,$\mathcal{C}$) —see Lemma 4.46 below. Although strictly akin to nwf_Unify_sets, nwf_Unify_sets$_*$ will not be guaranteed to terminate. The only differences between the two procedures are:

- $\mathcal{C}$ is regarded as a new input parameter. Therefore, it shall not be initialized to $\emptyset$ inside nwf_Unify_sets$_*$;

- in actions (3), (4), and (b.4), situations where nwf_Unify_sets places a new pair into $\mathcal{C}$, nwf_Unify_sets$_*$ *is free to do the same or to leave $\mathcal{C}$ unchanged*. (Note that the only remaining action that may affect $\mathcal{C}$, which is action (5), remains as before.)

Preliminary to stating the generalization of Theorem 4.44, we need a few definitions.

Definition 4.45 *A* Unify-tree *is a directed unordered, non-empty tree, each of whose nodes ν bears labels $\mathcal{E}_\nu, \mathcal{C}_\nu$ meeting the following conditions:*

- *$\mathcal{E}_\nu$ is a flat Herbrand system;*
- *$\mathcal{C}_\nu$ is a collection of literals of the form $Y \notin Q$, with Y and Q variables.*
- *A relationship reflecting the behavior of* nwf_Unify_sets$_*$ *holds between $\mathcal{E}_\nu, \mathcal{C}_\nu$ on the one hand and the tuple $\mathcal{E}_{\mu_1}, \mathcal{C}_{\mu_1}, \dots, \mathcal{E}_{\mu_p}, \mathcal{C}_{\mu_p}$, where $\mu_1, \dots, \mu_p$ are the distinct childrens of ν, on the other. To describe this relationship simply, it will be helpful to regard the identity $\mathcal{C} = \bigwedge_{Y\ in\ not\_in\{Q\}} Y \notin Q$ as an invariant rather than just as an initial condition. The relationship is as follows:*

  If the values of $\mathcal{E}$ and $\mathcal{C}$ are set to $\mathcal{E}_\nu$ and $\mathcal{C}_\nu$ at the beginning of a phase, then, depending on the first action that will affect either of them in the subsequent execution of nwf_Unify_sets$_*$*, one has that:*

  (1), (2), (3), (4), (5), (a): $p = 1$ and $\mathcal{E}_{\mu_1}$ and $\mathcal{C}_{\mu_1}$ are possible values of $\mathcal{E}$, $\mathcal{C}$ after the execution of the modifying action;

  (b): $p = 4$ and $\mathcal{E}_{\mu_i}$ and $\mathcal{C}_{\mu_i}$ are possible values of $\mathcal{E}$, $\mathcal{C}$ after the execution of ($b.i$);

  none: $p = 0$. In this case, ν will be called a failure *or a* success *leaf in agreement with the kind of exit that takes place (cf. actions* Fail_1*,* Fail_2*, and* Succeed*).*

A fringe *of a Unify-tree is a set S of nodes such that: every maximal path of the tree contains at most one node from S, and, if it contains none, it ends with a failure leaf.*

It should be clear from this definition that for any given pair $\mathcal{E}_*, \mathcal{C}_*$, a Unify-tree whose root is labeled $\mathcal{E}_*, \mathcal{C}_*$ exists. Unify-trees correspond in fact to the parallel executions of the algorithm $\mathsf{nwf\_Unify\_sets}_*(\mathcal{E}_*, \mathcal{C}_*)$: there are two sources of parallelism in $\mathsf{nwf\_Unify\_sets}_*$, namely action (b) and some freedom in putting literals into *not_in*. However, it is only the branching caused by action (b) that gets represented by a single Unify-tree.

Among others, a Unify-tree originates from the specific execution of $\mathsf{nwf\_Unify\_sets}_*$ that is entirely alike to an execution of $\mathsf{nwf\_Unify\_sets}$, except for the different initialization of *not_in*. In this execution, *not_in* literals are added whenever possible; moreover, the Unify-tree will be finite in this special case (cf. Theorem 4.34), and its set of success leaves will constitute a fringe.

Lemma 4.46 *Let $\mathcal{T}$ be a Unify-tree, with root ϱ. Then, for any fringe S of $\mathcal{T}$, the following holds:*

$$\texttt{NWF\_sets} \vdash (\mathcal{E}_\varrho \wedge \mathcal{C}_\varrho) \leftrightarrow \exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S} (\mathcal{E}_\nu \wedge \mathcal{C}_\nu)\ ,$$

where $Q_1, \ldots, Q_m$ are all the variables in the $\mathcal{E}_\nu$'s that do not occur in $\mathcal{E}_\varrho \wedge \mathcal{C}_\varrho$.

Proof. To prove the thesis, since

$\texttt{NWF\_sets} \vdash \neg(\mathcal{E}_\nu \wedge \mathcal{C}_\mu)$

trivially holds for every failure leaf μ, it will suffice to check that every single action of the algorithm $\mathsf{nwf\_Unify\_sets}_*$ leads from a node ν to nodes $\mu_1, \cdots, \mu_p$ such that

$\texttt{NWF\_sets} \vdash \mathcal{E}_\nu \wedge \mathcal{C}_\nu) \leftrightarrow \exists Z_1 \cdots \exists Z_\ell \bigvee_{i=1}^{p} (\mathcal{E}_{\mu_i} \wedge \mathcal{C}_{\mu_i}\ ,$

where $Z_1, \ldots, Z_\ell$ are the new variables (if any). In the following, each number on the left indicates the action being analyzed.

(1) $X \doteq X$, being true by $(\doteq_1)$, can be discarded from $\mathcal{E}_\nu$.

(2) For any formula φ, $X \doteq Y \wedge \varphi(X,Y)$ is equivalent to $X \doteq Y \wedge \varphi(Y,Y)$ by $(\doteq_2)$.

(3) Viewed as a formula, a zipper $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_{n-1}}{\leftarrow} X_n \overset{Y_n}{\leftarrow} X_0$ yields $X_i = X_0$, hence $Y_i \in X_0$, for $i = 0, \ldots, n$ —by (E^s_k) and (W). Accordingly, by assigning X_0 `less` Y_i to K_i,[6] one has both $X_0 \doteq \{Y_i \,|\, K_i\}$ and $Y_i \notin K_i$ for all i—by (W) and (L).

It follows that

$$(W)(E^s_k)(L) \vdash (\mathcal{E}_\nu \wedge \mathcal{C}_\nu) \to \exists K_0 \cdots \exists K_n \left(\mathcal{E}_{\mu_1} \wedge \mathcal{C}_\nu \wedge \bigwedge_{i=0}^{n} Y_i \notin K_i \right)$$

Conversely, as is easily seen,

$$(W)(\doteq) \vdash (\mathcal{E}_{\mu_1} \wedge \mathcal{C}_\nu) \to (\mathcal{E}_\nu \wedge \mathcal{C}_\nu)\,.$$

(4) From $X_0 \overset{Y_0}{\leftarrow} X_1 \overset{Y_1}{\leftarrow} \cdots \overset{Y_n}{\leftarrow} X_{n+1} \overset{Y_0}{\leftarrow} X_{n+2}$ by (W) and (E^s_k), it follows that $X_0 \doteq X_1$. In fact, $Y_0 \in X_{n+1}$, whence $Y_0 \in X_n, \cdots, Y_0 \in X_1$. Hence, the insertion, of Y_0 into X_1 has no effect and, thanks to $(\doteq)$, the equality $X_0 \doteq \{Y_0 \,|\, X_1\}$ can be simplified into $X_0 \doteq X_1$.

(5) $\mathcal{E}_\nu \wedge \mathcal{C}_\nu$ is equivalent to $\mathcal{E}_{\mu_1} \wedge \mathcal{C}_{\mu_1}$ by virtue of (W) alone.

(a) $X \doteq f(X_1, \ldots, X_n) \wedge X \doteq f(Y_1, \ldots, Y_n)$ is equivalent to $X \doteq f(X_1, \ldots, X_n) \wedge X_1 \doteq Y_1 \wedge \cdots \wedge X_n \doteq Y_n$. One direction follows from (F_1), and the other from $(\doteq_2)$.

(b) (E^s_k) states that $e \equiv X \doteq \{Y \,|\, V\} \wedge e' \equiv X \doteq \{Z \,|\, W\}$ if and only if

- $\varphi_1 \equiv e \wedge e' \wedge Y \doteq Z \wedge V \doteq W$, or
- $\varphi_2 \equiv e \wedge e' \wedge X \doteq \{Z \,|\, W\} \wedge Y \doteq Z \wedge V \doteq X$, or
- $\varphi_3 \equiv e \wedge e' \wedge X \doteq \{Y \,|\, V\} \wedge Y \doteq Z \wedge W \doteq X$, or
- $\varphi_4 \equiv e \wedge e' \wedge \exists N\, (V \doteq \{Z \,|\, N\} \wedge W \doteq \{Y \,|\, N\})$.

Consider the formula φ_4. It can be split into 4 disjuncts:

- $\varphi_{4.1} \equiv Y \in N \wedge Z \in N \wedge \varphi_4$,

[6]Since we are dealing with finite structure only, the `less` operation can be constructively implemented—cf. Lemma 3.18, which holds also for sets.

- $\varphi_{4.2} \equiv Y \in N \wedge Z \notin N \wedge \varphi_4$,
- $\varphi_{4.3} \equiv Y \notin N \wedge Z \in N \wedge \varphi_4$,
- $\varphi_{4.4} \equiv Y \notin N \wedge Z \notin N \wedge \varphi_4$.

$\varphi_{4.4}$ is exactly the formula of action (b.4) (the equation e can be deduced).

From $\varphi_{4.1}$ it is immediate to check that $X = V = W = N$, hence it is an instance of both th formula corresponding to action (b.2) and of action (b.3).

$\varphi_{4.2}$ implies, in particular, that $Y \neq Z$, and that $W \doteq \{Y \mid N\}$ and $Y \in N$, namely, $N = W$. This implies that $X = V$. Hence it can be re-written as $Y \neq Z \wedge X \doteq V \wedge e \wedge e'$. Such case can be combined with the formula φ_2 above allows to obtain exactly the formula corresponding to action (b.2).

The relations between $\varphi_{4.3}$, φ_3, and the formula corresponding to action (b.3) follows in a similar way.

To end, notice that the formula φ_1 is exactly he formula corresponding to action (b.1).

4.46 □

Proof of Theorem 4.44 Let us consider the Unify-trees $\mathcal{T}_0, \mathcal{T}_1$ corresponding to the executions of $\mathsf{nwf\_Unify\_sets}_*(\mathcal{E}, \emptyset)$ that respectively add *not_in* literals

- whenever possible,
- never.

$\mathcal{T}_0$ corresponds to the execution of $\mathsf{nwf\_Unify\_sets}(\mathcal{E})$; hence it is finite, by Theorem 4.34. The preceding lemma, referred to the fringe S_0 consisting of all success leaves of $\mathcal{T}_0$, gives

$$\mathcal{E} \leftrightarrow \exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S_0} (\mathcal{E}_\nu \wedge \mathcal{C}_\nu) .$$

$\mathcal{T}_1$ clearly contains an isomorphic copy $\mathcal{T}_0'$ of $\mathcal{T}_0$: every node μ in $\mathcal{T}_0$ is labeled $\mathcal{E}_\mu, \mathcal{C}_\mu$, while the corresponding node μ' is labeled $\mathcal{E}_\mu, \emptyset$ in $\mathcal{T}_0'$.

We consider the fringe S_1 of $\mathcal{T}_1$ consisting of all nodes ν' such that ν either belongs to S_0 or is a Fail_1 leaf of $\mathcal{T}_0$. Again by Lemma 4.46, we have

$$\mathcal{E} \leftrightarrow \exists Q_1 \cdots \exists Q_m \exists Q_{m+1} \cdots \exists Q_{m+k} \bigvee_{\nu' \text{ in } S_1} \mathcal{E}_\nu \, .$$

Assuming $\exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S_0} \mathcal{E}_\nu$, one readily obtains

$\exists Q_1 \cdots \exists Q_{m+k} \bigvee_{\nu' \text{ in } S_1} \mathcal{E}_\nu$,

whence $\mathcal{E}$ follows. Conversely, from $\mathcal{E}$, one derives

$\exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S_0}(\mathcal{E}_\nu \wedge \mathcal{C}_\nu)$,

which entails

$\exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S_0} \mathcal{E}_\nu$.

We conclude with the desired thesis: $\mathcal{E} \leftrightarrow \exists Q_1 \cdots \exists Q_m \bigvee_{\nu \text{ in } S_0} \mathcal{E}_\nu$.
4.44 □

4.3.4 NWF-bag unification

In § 4.3.2 and in § 4.3.3 we have presented two unification algorithms based on a two-level '`repeat`'. The same architecture will be used here to develop the unification algorithm for the non-well-founded theory of hybrid (even infinite) multi-sets `NWF_sets` presented in § 3.2.2.

As anticipated in § 4.3, explicit solvable form systems (cf. Def. 4.18) admits *finite* bags solutions. On the contrary, if a subsystem contains a *zipper* (cf. Def. 3.38):

$$X_0 \doteq \{\!\![\, Y_0 \,|\, X_1 \,]\!\!\}, \ldots, X_{m-1} \doteq \{\!\![\, Y_{m-1} \,|\, X_m \,]\!\!\}, X_m \doteq \{\!\![\, Y_m \,|\, X_0 \,]\!\!\}$$

(for $m = 0$ this reduces to the single equation $X_0 \doteq \{\!\![\, Y_0 \,|\, X_0 \,]\!\!\}$), this means that

$$X_0 = X_1 = \cdots = X_m$$

and that $Y_0, \ldots, Y_m$ all belong to the bag X_0 an infinite number of times. The handling of such situations is performed by action (3).

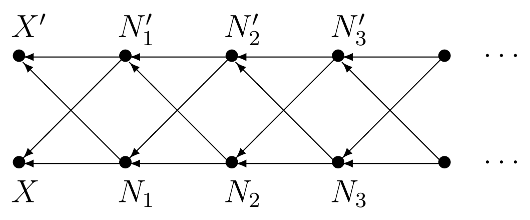
When two bags are to be unified (namely, when two equations $X \doteq \{\!\![\, Y \,|\, V \,]\!\!\}$, and $X \doteq \{\!\![\, Z \,|\, W \,]\!\!\}$ are both in $\mathcal{E}$—assume that V and W are distinct from X) one of the two situations

(1) $Y \doteq Z \wedge V \doteq W$
(2) $\exists N\,(V \doteq \{\!\!\{\, Z \mid N \,\}\!\!\} \wedge W \doteq \{\!\!\{\, Y \mid N \,\}\!\!\})$

should hold (cf. axiom (E^m_k)—§ 3.1.2). However, if such rule is implemented in the following way, directly suggested by the axiomatization:

$X \doteq \{\!\!\{\, Y \mid V \,\}\!\!\} \wedge X \doteq \{\!\!\{\, Z \mid W \,\}\!\!\} \wedge \mathcal{E}$
return non-deterministically:
(a_1) $Y \doteq Z \wedge V \doteq W \wedge X \doteq \{\!\!\{\, Z \mid W \,\}\!\!\} \wedge \mathcal{E}$,
(a_2) $X \doteq \{\!\!\{\, Y \mid V \,\}\!\!\} \wedge V \doteq \{\!\!\{\, Z \mid N \,\}\!\!\} \wedge W \doteq \{\!\!\{\, Y \mid N \,\}\!\!\} \wedge \mathcal{E}$,

it is easy to find non-terminating sequences of actions (cf. with a similar problem in § 4.2.2). For instance, if the input system is $X \doteq \{\!\!\{\, Y \mid V \,\}\!\!\}, X \doteq \{\!\!\{\, Z \mid W \,\}\!\!\}, X' \doteq \{\!\!\{\, Y' \mid V \,\}\!\!\}, X' \doteq \{\!\!\{\, Z' \mid W \,\}\!\!\}$, then applying always action (a_2), the situation schematically represented in the following diagram may be generated (Y, Y', Z, Z' are not influential for the computation, hence they are omitted in the diagram):



In the case of (hybrid) hypersets (cf. § 4.3.3), the technical way to escape such situations is to check whether N represents the set V deprived of Z (and the set W deprived of Y). This check is performed by action Fail_2, using the data-structure $\mathcal{C}$.

In the case of bags, it is no longer possible to perform such kind of check, since one element can be removed from a bag an arbitrary number (even infinite!) of times generating always different bags.

What we will prove is that, even it is perfectly correct to find solutions generating chains of bag inclusions of arbitrarily length, only a finite number of removing is sufficient to gain completeness of the search tree (the other computations would look for instances of already computed solutions). This allows to force the pruning of the search tree, and to put a polynomial bound on the length of any non-deterministic computation.

To begin with, we prove the following combinatorial lemma, whose intuitive meaning is that if there exists a solution to a system of equations, then there exits a simpler solution to it. Moreover, the number of *finite* occurrences of elements in a bag in the simpler solution can be bounded by the number of occurrences of the bag constructor symbol in the system.

Lemma 4.47 *Let $\mathcal{E}$ consisting of*

- *ℓ equations of the form $X_i \doteq \{\!\!\{ X_j \mid X_k \}\!\!\}$ and*
- *a certain number of equations of the form $X_i \doteq \{\!\!\{ \}\!\!\}$.*

Let $X_1, \ldots, X_m$ be the variables occurring in $\mathcal{E}$. Then, for any solution θ to $\mathcal{E}$ in a model of bag theory, there exists a ground solution θ' to $\mathcal{E}$ such that, for every $i = 1, \ldots, m$, the following conditions hold:

1. *for all $j = 1, \ldots, m$, if $X_i\theta = X_j\theta$, then $X_i\theta' = X_j\theta'$;*
2. *for all $j = 1, \ldots, m$, one of the following holds:*
   - *(a) $X_j\theta' \in^n X_i\theta'$, where $n \leq \ell$, or*
   - *(b) $X_j\theta' \in^\omega X_i\theta'$;*
3. *if $Y \in^\alpha X_i\theta'$, then $Y = X_j\theta'$, for some $j = 1, \ldots, m$.*

Proof. Let θ be a solution to $\mathcal{E}$ in a model of bag theory; for $i = 1, \ldots, m$, let $a_i = X_i\theta$. The following preliminary definitions are useful in the remaining part of the proof.

- θ induces a congruence relation $\approx$ over $X_1, \ldots, X_m$: $X_i \approx X_j$ if and only if $a_i = a_j$.
- $\approx$ can be used to determine another equivalence relation $\sim$ over the same set of variables defined as the transitive and reflexive closure of the following rule:

  $X_i \sim X_j$ if there is an equation in $\mathcal{E}$ of the form $X \doteq \{\!\!\{ \cdot \mid V \}\!\!\}$ such that $X_i \approx X$ and $X_j \approx V$.

- For all i and j, we define

$$\mu_i^j \quad = \quad \begin{cases} \omega & \text{if } a_j \in^\alpha a_i,\ \alpha \geq \omega \\ \min_{k:X_i \sim X_k}\{h : a_j \in^h a_k\} & \text{otherwise} \end{cases}$$

  Observe that, if $X_i = \{\!\!\{ X_j \,|\, X_k \}\!\!\}$ belongs to $\mathcal{E}$, then $\mu_i^j = \mu_k^j$.

We compute the function '$*$' using the following algorithm:

```
$a_i^* := \{\!\!\{ \ \}\!\!\}$;
for $j = 1$ to $m$ do
    if $X_i \sim X_j$ and $a_j$ is distinct from $a_1, \dots, a_{j-1}$
    then $a_i^* := a_i^* \uplus \begin{cases} \{\!\!\{ \underbrace{a_j^*, a_j^*, \dots}_{\omega} \}\!\!\} & \text{if } a_j \in^\alpha a_i, \omega \leq \alpha \\ \{\!\!\{ \underbrace{a_j^*, \dots, a_j^*}_{h} \}\!\!\} & \text{if } a_j \in^{h+\mu_i^j} a_i, h + \mu_i^j \in \omega \end{cases}$
```

Since two bags represented by X_i and X_k, where $X_i \sim X_k$, can differ for (at most) ℓ occurrences of elements, then, for any i and j, $a_j^* \in^\alpha a_i^*$ for $\alpha = \omega$ or for $\alpha \leq \ell$.

Hence, the three conditions trivially hold for the substitution $\theta' = [X_1/a_1^*, \dots, X_m/a_m^*]$. It remains to show that it is also a (ground) solution to $\mathcal{E}$. It is sufficient to verify that, for any equation $X_i \doteq \{\!\!\{ X_j \,|\, X_k \}\!\!\}$, $a_i^* = \{\!\!\{ a_j^* \,|\, a_k^* \}\!\!\}$.

By assumption $a_i = \{\!\!\{ a_j \,|\, a_k \}\!\!\}$; assume first that $a_i = a_k$. This implies that $a_i^* = a_k^*$ and $a_j^* \in^\omega a_i^* = a_k^*$. This ensures that $a_i^* = \{\!\!\{ a_j^* \,|\, a_k^* \}\!\!\}$.

Assume now that $a_i \neq a_k$. This implies that $a_j \in^h a_k$ and $a_j \in^{h+1} a_i$. By construction of the function $*$, we have that $a_j^* \in^{h+1-\mu_i^j} a_i^*$ and $a_j^* \in^{h-\mu_i^j} a_k^*$. It only remains to show that, for any other element a of a_i^* and a_k^*, we have

$$\begin{array}{llll} a \in^\omega a_i^* & \text{if and only if} & a \in^\omega a_k^* & \text{and} \\ a \in^h a_i^* & \text{if and only if} & a \in^h a_k^* & h \leq \ell\,. \end{array}$$

This fact follows by definition of the function $*$ and by the fact that $a_i = \{\!\!\{ a_j \,|\, a_k \}\!\!\}$.

4.47 □

In other words, any solution for a variable X defined into a an equation system $\mathcal{E}$ (involving the functional symbols for bags this thesis deals with) can be finitely represented in this way:

$$\begin{array}{l} X \doteq \{\!\{ Y_1 \,|\, X \}\!\}, \dots, X \doteq \{\!\{ Y_p \,|\, X \}\!\}, \\ X \doteq \{\!\{ Z_1 \,|\, V_1 \}\!\}, V_1 \doteq \{\!\{ Z_2 \,|\, V_2 \}\!\}, \dots, V_{n-1} \doteq \{\!\{ Z_n \,|\, V_n \}\!\}, \\ \text{(possibly) } V_n \doteq \{\!\{ \ \}\!\} \end{array}$$

where

- $Y_1, \dots, Y_p$ are distinct elements that belong an infinite (ω) number of times to X;
- $Z_1, \dots, Z_n$, all distinct from $Y_1, \dots, Y_p$, belong to it finitely.

The lemma ensures also that $n \leq \ell$, where ℓ is the number of occurrences of the functional symbol $\{\!\{ \cdot \,|\, \cdot \}\!\}$ in $\mathcal{E}$.

Such a bound can be further refined as suggested by the proof of the Lemma 4.47. Every equation occurring in a *zipper* predicates only on infinite membership. This means that the number of initial equations that will occur in a zipper during the computation can be subtracted from ℓ to place a stricter bound to the length of 'useful' paths.

All the above considerations continue to hold for variable denoting bags even when in the system free functional symbols occur. We only need to be sure that the pruning of a tree will not result in a loss of completeness. If the *kernel* of two bags unify, the algorithm behaves as if they were both $\{\!\{ \ \}\!\}$; otherwise the two bags do not unify, hence the pruning does not result in a loss of completeness.

This stream of ideas will have an immediate counter-part in the algorithm we are going to present. It is sufficient that the length m of any path in $\mathcal{E}$

$$X_0 \overset{Y_1}{\leftarrow} X_1 \overset{Y_2}{\leftarrow} \dots \overset{Y_{m-1}}{\leftarrow} X_{m-1} \overset{Y_m}{\leftarrow} X_m$$

(X_i pairwise distinct) be bounded by $\ell - k$, where k is the number of occurrences of initial equations of the form $\_ \doteq \{\!\{ \_ | \_ \}\!\}$ used in 'zippers'.

The unification algorithm is presented in Fig. 4.8.

```
nwf_Unify_bags($\mathcal{E}$: Herbrand_system);
let $\ell$ be the number of occurrences of the functional symbol $\{\!\{\cdot\,|\,\cdot\}\!\}$ in $\mathcal{E}$;
$k := 0$;
repeat
Preamble: repeat
    (1)    $X \doteq X \wedge \mathcal{E} \mapsto \mathcal{E}$
    (2)    $\left.\begin{array}{c} X \doteq Y \wedge \mathcal{E} \\ X \text{ occurs in } \mathcal{E} \end{array}\right\} \mapsto \mathcal{E}[X/Y] \wedge X \doteq Y$
    (3)    $\left.\begin{array}{c} \text{if there is a zipper } X_0 \doteq \{\!\{ Y_0 \,|\, X_1 \}\!\}, \dots, \\ X_{m-1} \doteq \{\!\{ Y_{m-1} \,|\, X_m \}\!\}, X_m \doteq \{\!\{ Y_m \,|\, X_0 \}\!\} \\ m > 0,\ X_0, \dots, X_m \text{ distinct from one another} \end{array}\right\} \mapsto$
               $\mathcal{E} \wedge X_1 \doteq X_0 \wedge \dots \wedge X_m \doteq X_0$
               let $n$ be the number of initial edges from
               the $m+1$ considered; let $k := k + n$
until nothing has been modified by the last iteration;
Turning point:
  Fail_1: if there is a path of length greater than $\ell - k$ in $\mathcal{E}$
      then exit with failure;
  Fail_2: if there are equations $X \doteq f(X_1, \dots, X_n)$,
      and $X \doteq g(Y_1, \dots, Y_m)$ in $\mathcal{E}$ with $f \not\equiv g$, then exit with failure;
  Succeed: if $\mathcal{E}$ is in solvable form, then exit with success returning $\mathcal{E}$;
Actions: perform, non-deterministically, one of the following actions
    (a)  $X \doteq f(X_1, \dots, X_n) \wedge X \doteq f(Y_1, \dots, Y_n) \wedge \mathcal{E} \mapsto$
         $X \doteq f(Y_1, \dots, Y_n) \wedge \mathcal{E} \wedge X_1 \doteq Y_1 \wedge \dots \wedge X_n \doteq Y_n$
    (b)  $\left.\begin{array}{r} X \doteq \{\!\{ Y \,|\, X \}\!\} \wedge X \doteq \{\!\{ Z \,|\, W \}\!\} \wedge \mathcal{E} \\ X \not\equiv W \end{array}\right\} \mapsto$
         (b.1)  $X \doteq \{\!\{ Y \,|\, X \}\!\} \wedge W \doteq X \wedge Y \doteq Z \wedge \mathcal{E}$
                $(Z \in^{\omega} X = W)$
         (b.2)  $W \doteq \{\!\{ Y \,|\, W \}\!\} \wedge X \doteq \{\!\{ Z \,|\, W \}\!\} \wedge \mathcal{E}$
                $(Z \in^{h} W, Z \in^{h+1} X)$
    (c)  $\left.\begin{array}{r} X \doteq \{\!\{ Y \,|\, V \}\!\} \wedge X \doteq \{\!\{ Z \,|\, W \}\!\} \wedge \mathcal{E} \\ X \not\equiv V \text{ and } X \not\equiv W \end{array}\right\} \mapsto$
         (c.1)  $X \doteq \{\!\{ Z \,|\, W \}\!\} \wedge V \doteq W \wedge Y \doteq Z \wedge \mathcal{E}$
         (c.2)  if $V \not\equiv W$ then $X \doteq \{\!\{ Y \,|\, V \}\!\} \wedge V \doteq \{\!\{ Z \,|\, N \}\!\} \wedge$
                    $W \doteq \{\!\{ Y \,|\, N \}\!\} \wedge \mathcal{E}$
forever.
```

Figure 4.8: Hyperbag unification algorithm

If only finite bags solutions are required, then remove action (b) and replace the updating performed by action (3) with `fail`.

The proof of termination and of the fact that the algorithm shows that the non well-founded bags unification problem belongs to NP are very similar (but simpler) to the corresponding proofs for the (hybrid) hyperset case (cf. § 4.3.3). More in detail,

Theorem 4.48 (Termination and Complexity) $\mathsf{nwf\_Unify\_bags}(\mathcal{E})$ *always terminates. Moreover, if* v_0 *and* s_0 *are the number of variables and of occurrences of functional symbols in* $\mathcal{E}$*, respectively, then in any non-deterministic computation the number of executed actions is polynomially bound by* $(v_0 + s_0)$*.*

Proof. First note that if action (c.2) is not executed, then at any phase one occurrence of functional symbol disappear from the system. This is clear in the case of (a), (b.1), and (c.1). Action (c.2) apparently does not cause such complexity decreasing; however, the equation $X \doteq \{\!\![\, Z \mid W \,]\!\!\}$ is the only one in which X does occur, thus, it will be never considered again by the algorithm.

Then we observe that a preamble can at most perform $O(v + s)$ actions, where v and s are the the number of variables and of occurrences of functional symbols in $\mathcal{E}$ at the beginning of the preamble considered.

To conclude the proof, it only remains to show that only a polynomial number of variables can be generated in a non-deterministic computation. Observe that, if a generated edge occur in a zipper, then it cannot generate any new edge and variable (the generation power of the algorithm decreases). Moreover, when this occurs, the data-structure action (3) updates k to keep into account this fact, putting a stricter limit to the length of the paths. Thus, the combinatorial Theorem 4.42 developed for hyperset continues to hold, (the bound limit for paths is given by $\ell \leq s_0$) ensuring that the number of generated variables is $O((s_0)^3)$.

4.48 □

Soundness is straightforward, for completeness, we need to call into play the combinatorial lemma 4.47.

Theorem 4.49 (Soundness and Completeness) *Let $\mathcal{E}_1, \dots, \mathcal{E}_n$ be the systems non-deterministically returned by* $\mathsf{nwf\_Unify\_bags}(\mathcal{E})$ *($n = 0$ means that all the computations end with* `fail`*). Then*

$$\texttt{NWF\_bags} \vdash \mathcal{E} \leftrightarrow \exists N_1 \cdots N_h \bigvee_{i=1}^{n} \mathcal{E}_i,$$

where $N_1, \dots, N_h$ are the variables occurring in $\mathcal{E}_1, \dots, \mathcal{E}_n$ but not in $\mathcal{E}$.

Proof. The proof develops in two parts. In the first part we show that any action distinct from $\mathsf{Fail\_1}$ locally guarantees the claim. Later, using the combinatorial Lemma 4.47, we will show that $\mathsf{Fail\_1}$ does not force a loss of completeness. The finiteness of each computation, ensured by Theorem 4.48, concludes the proof.

Soundness and completeness of actions (1) and (2) follows trivially by ($\doteq$). One direction of actions (3) follows from (E_k^m). The other is trivial. $\mathsf{Fail\_1}$ action is justified by Lemma 4.47. Action (a) is justified in one sense by ($\doteq$), in the other by axiom (F_1). Action (c) is exactly the implementation of axiom (E_k^m).

Let us analyze action (b). Axiom (E_k^m) states that $X \doteq \{\!\!\{ Y \mid X \}\!\!\} \wedge X \doteq \{\!\!\{ Z \mid W \}\!\!\}$ is equivalent to the disjunction

$$\begin{array}{ll} (1) & (X \doteq \{\!\!\{ Y \mid X \}\!\!\} \wedge Y \doteq Z \wedge W \doteq X) \vee \\ (2) & X \doteq \{\!\!\{ Z \mid W \}\!\!\} \wedge \exists N \, (X \doteq \{\!\!\{ Z \mid N \}\!\!\} \wedge W \doteq \{\!\!\{ Y \mid N \}\!\!\}) \end{array}$$

The disjunct (1) is exactly the formula introduced by action ($b.1$). Since (E_k^m) states, in particular, that $X \doteq moZ \mid W \}\!\!\} \wedge X \doteq \{\!\!\{ Z \mid N \}\!\!\}$ implies $W \doteq N$, the second disjunct is equivalent to the formula introduced by action ($b.2$).

It remains to see that the pruning of the search tree performed by action $\mathsf{Fail\_1}$ in the $\mathsf{Turning\ point}$ does not cause a loss of completeness. The algorithm, in a branch in which more than $\ell - k$ variables have been generated, looks for a solution that, in view of Lemma 4.47, it can find with less than $\ell - k$ removing of elements. Hence, such branch is superfluous.

4.49 □

4.4 A minimality analysis

Experience in programming with sets teachs that the number of unifiers returned by a set unification algorithm is, in general, very large. Therefore, one of the main goals of a unification algorithm is to reduce, as much as possible, redundancies in the set of solutions returned. Clearly, the best situations is that a unification algorithm for a theory T is *minimal* for it (cf. Def. 2.13), namely, that it returns exactly the minimal complete set of unifiers, without repetitions, without instances, for any unification problem.

Given a unification problem between two sets it is possible, in principle, to determine the minimal number of an independent set of unifiers of maximal generality for it (i.e. the minimum cardinality of a $\mu \bigcup_{(E_1)(E_2)}(s,t)$—cf. § 2.2). Nevertheless, it is impossible to experimentally test the *minimality* (namely the capability of returning exactly $\mid \mu \bigcup_{(E_1)(E_2)}(s,t) \mid$ solutions) for all possible (infinite) problems $s \doteq t$ with s,t in $\tau(\Sigma \cup \mathcal{V})$. We propose a number of sample goals on which the minimality of an algorithm can be tested, as done in the famous paper [21] in the contect of AC-unification. Such choice is suggested by the following reasoning:

- problems with *nested* sets (i.e. sets containing sets, such as example above) give rise to confusion in the analysis. It is more important to concentrate the efforts in pointing out the new unification problems between elements of the two sets that must be generated. This can be seen as the reduction from a problem to a set of problem of fewer size; in other words, it allows us to analyze the minimality of the induction step;

- *closed* sets (i.e. of the form $\{t_1 \mid \cdots \{t_n \mid \emptyset\} \cdots\}$) and *open* sets (ending in a variable, as the following $\{t_1 \mid \cdots \{t_n \mid R\} \cdots\}$) can be described. Any unification problem between two sets can be in one of the following forms:

  - closed with closed (problems (1)–(4));
  - open with closed (or *vice-versa*) (problems (5) and (6));
  - open with open (problems (7) and (8)).

In each of such cases, if the elements of the sets are distinct variables, the number of possible solutions is maximized (for instance $\{X_1, X_2\} \doteq \{Y_1, Y_2\}$ admits more solutions either than $\{X_1, X_2\} \doteq \{Y_1, X_2\}$ or than $\{X_1, X_2\} \doteq \{a_1, a_2\}$).

As particular case, it is interesting to analize the cleverness of an algorithm in solving the problem in which the two sets share elements (problems (3) and (4)). Moreover, also a shrewd treatment of the *matching* problem (namely when one of the two sets is ground) is important (problems (1) and (5)).

The unification problem between open sets must be analyzed differently whether the 'rest' variables are identical or not. The announced sample problems are the following:

$$\begin{array}{lrcl}
(1) & \{X_1,\dots,X_m\} & \doteq & \{a_1,\dots,a_n\}\,, \\
(2) & \{X_1,\dots,X_m\} & \doteq & \{Y_1,\dots,Y_n\}\,, \\
(3) & \{X_1,\dots,X_m,a_1,\dots,a_k\} & \doteq & \{Y_1,\dots,Y_n,a_1,\dots,a_k\}\,, \\
(4) & \{X_1,\dots,X_m,Z_1,\dots,Z_k\} & \doteq & \{Y_1,\dots,Y_n,Z_1,\dots,Z_k\}\,, \\
(5) & \{X_1,\dots,X_m \,|\, Z\} & \doteq & \{a_1,\dots,a_n\}\,, \\
(6) & \{X_1,\dots,X_m \,|\, Z\} & \doteq & \{Y_1,\dots,Y_n\}\,, \\
(7) & \{X_1,\dots,X_m \,|\, Z\} & \doteq & \{Y_1,\dots,Y_n \,|\, Z\}\,, \\
(8) & \{X_1,\dots,X_m \,|\, W\} & \doteq & \{Y_1,\dots,Y_n \,|\, Z\}\,,
\end{array}$$

where

- $X_1,\dots,X_m,Y_1,\dots,Y_n,Z_1,\dots,Z_k,W,Z$ are pairwise distinct variables, and

- $a_1,\dots,a_{\max\{n,k\}}$ are pairwise distinct constant symbols.

Remark 4.50 *It is important to notice that problems* (1)*,* (6)*,* (7)*, and* (8) *constitute a complete set of templates into which* <u>any</u> *set unification problem written in the signature* $\{\emptyset, \{\cdot\,|\,\cdot\}, \dots\}$ *is comprised. For instance, any solution* θ *to problem* (2)*—which consists of a number of mappings of the form* X_i/Y_j *or* Y_j/X_i*—can be seen as a* template *useful to generate a number of equations between the elements* $s_1,\dots,s_m$ *and* $t_1,\dots,t_n$ *of the two generic sets in the equation* $\{s_1,\dots,s_m\} \doteq$

$\{t_1, \ldots, t_n\}$. *More precisely, the latter unification problem can be reduced to the conjunction*

$$\bigwedge_{X_i/Y_j \; in \; \theta \vee Y_j/X_i \; in \; \theta} s_i \doteq t_j \; .$$

In § 4.4.1 we will describe the functions which compute the minimum cardinality of a $\mu \bigcup_{(E_1)(E_2)}$ for problems (1)–(8) (tables reporting some values for them are presented in § A.6). The behavior of the algorithm Unify_sets (cf. § 4.2.4) with respect to the eight sample problems is the argument of 4.4.2. Finally, in § 4.4.3 we present a unification algorithm, named SUA, which is optimal for all problems (1)–(8).

4.4.1 Sample problems

In this section we will count, from a combinatorial point of view, the minimal number of $(E_1)(E_2)$-unifiers that a $\mu \bigcup_{(E_1)(E_2)}(s, t)$, where $s \doteq t$ has one of the eight form above, must contain. To avoid proliferation of trivial lemmata, while the combinatorial analysis is carried on in all details, the fact that such solutions are $(E_1)(E_2)$-unifiers of $s \doteq t$ is left to the intuition of the reader.

The problem (1)

The number of solutions to the problem

$$(1) \qquad \{X_1, \ldots, X_m\} \;\doteq\; \{a_1, \ldots, a_n\}$$

is exactly the number of surjective applications from a set of m elements onto a set of n elements (such number will be denoted as $\mathit{Surj}(m, n)$).

If $m < n$ then $\mathit{Surj}(m, n) = 0$. Assume $m \geq n$; if $m = n = 0$ then $\mathit{Surj}(m, n) = 1$; if $m > 0$ and $n = 0$ then $\mathit{Surj}(m, n) = 0$. If $n = 1$, then $\mathit{Surj}(m, n) = 1$.

Let $n > 1$: any surjective function $g : \{X_1, \ldots, X_m\} \rightarrow \{a_1, \ldots, a_n\}$ can be obtained as follows:

- select a_j (n different ways of making such choice);

- select a nonempty subset S of $\{X_1, \ldots, X_m\}$ ($|S| = i+1$). Extend any surjective function

  $$g : \{X_1, \ldots, X_m\} \setminus S \to \{a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n\}$$

  so as to $g(X) = a_j$, for any $X \in S$. In order such g exists, $m - (i+1)$ cannot be less than $n - 1$, i.e. $i \leq m - n$.

The recursive definition for *Surj* when $m \geq n > 0$ will be

$$\begin{cases} \mathit{Surj}(m,1) = 1 & m \geq 1 \\ \mathit{Surj}(m,n) = n \sum_{i=0}^{m-n} \binom{m-1}{i} \mathit{Surj}(m-1-i, n-1) & m \geq n > 1 \end{cases}$$

Alternatively, any surjective function $g : \{X_1, \ldots, X_m\} \to \{a_1, \ldots, a_n\}$ can be obtained:

- extending a surjective function

  $$g : \{X_1, \ldots, X_{m-1}\} \to \{a_1, \ldots, a_n\}$$

  so as to $g(X_m) = a_i$ (there are n possibilities for choosing such i);

- extending uniquely a surjective function

  $$g : \{X_1, \ldots, X_{m-1}\} \to \{a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n\}$$

  (there are n possibilities for choosing such i), so as to fulfill $g(X_m) = a_i$.

This suggests the following recursive definition for the function *Surj*:

$$\begin{cases} \mathit{Surj}(m,n) &= 0 & \text{if } m < n \\ \mathit{Surj}(m,1) &= 1 & \text{if } 1 \leq m \\ \mathit{Surj}(m,n) &= n \cdot \left(\begin{array}{c} \mathit{Surj}(m-1, n-1)+ \\ \mathit{Surj}(m-1, n) \end{array} \right) & \text{if } 1 < n \leq m. \end{cases}$$

A more compact description of $\mathit{Surj}(m,n)$ can be given using *Stirling numbers* of the second type (see, for instance [61]): $\left\{ {m \atop k} \right\}$ is the number of ways to partition a set of m elements into k nonempty disjoint subsets. Any surjective function $g : \{X_1, \dots, X_m\} \to \{a_1, \dots, a_n\}$ can be obtained mapping (with a bijection) an n-partition of $\{X_1, \dots, X_m\}$ onto the set $\{a_1, \dots, a_n\}$. Thus,

$$Surj(m, n) = n! \left\{ {m \atop n} \right\}.$$

However, the recursive nature of the definition is not removed, but only hidden into the definition of *Stirling number*

The problem (2)

We will describe a function $\Phi : \omega^2 \to \omega$ computing the number of most general and independent solutions to the problem

$$(2) \qquad \{X_1, \dots, X_m\} \doteq \{Y_1, \dots, Y_n\}.$$

If $m = 0$ and $n = 0$ then $\Phi(m, n) = 1$; if $m > 0$ and $n = 0$, or $m = 0$ and $n > 0$, then $\Phi(m, n) = 0$.

Assume $m, n > 0$; if $m = 1$ or $n = 1$ then $\Phi(m, n) = 1$; otherwise (i.e. when $m, n > 1$) fix an element in the first set, say X_m. Two cases must be analyzed:

- X_m is joined to all elements of a subset S of $\{X_1, \dots, X_{m-1}\}$ and mapped onto one element Y_i, for some $i = 1, \dots, n$ (n ways). This mapping is then added to any solution to the sub-problem $\{X_1, \dots, X_{m-1}\} \setminus S \doteq \{Y_1, \dots, Y_{i-1}, Y_{i+1}, \dots, Y_n\}$;
- X_m is mapped to all elements of a subset T of $\{Y_1, \dots, Y_n\}$ such that $|T| \geq 2$ (if $|T| = 1$ then it would be one of the cases analyzed in the previous item). This mapping is then added to any solution to the sub-problem $\{X_1, \dots, X_{m-1}\} \doteq \{Y_1, \dots, Y_n\} \setminus T$.

Pick S nonempty (former case) and T such that $|T| \geq 2$ (latter case). Select $Y_j \in T$ and consider the solution $\theta = [X_m/Y_j] \cup [X_i/Y_j : X_i \in S] \cup [Y_p/Y_j : Y_p \in T, j \neq p] \cup \theta_1$, where θ_1 is a solution to $\{X_1, \dots, X_{m-1}\} \setminus S \doteq \{Y_1, \dots, Y_n\} \setminus T$. This solution is not faced by one of the two cases above. However, for any $Y_p \in T$, $p \neq j$, θ is an instance of the solution $\theta' = [X_m/Y_j] \cup [X_i/Y_p : X_i \in S] \cup [Y_\ell \doteq Y_p : Y_\ell \in T, \ell \neq j, p]$. This process can be iterated until in the solution there are no situations of the described form.

For $m, n > 0$ the function Φ will be recursively described as follows:

$$\begin{cases} \Phi(m,1) &= 1 & m \geq 1 \\ \Phi(1,n) &= 1 & n > 1 \\ \Phi(m,n) &= n \sum_{i=0}^{m-2} \binom{m-1}{i} \Phi(m-1-i, n-1) + & m,n > 1 \\ & \quad \sum_{j=2}^{n-1} \binom{n}{j} \Phi(m-1, n-j)\,. & \end{cases}$$

The following alternative analysis of Φ will be useful for the study of problems (6) and (8).

From the above description it follows that any most general unifier θ of $\{X_1, \ldots, X_m\} \doteq \{Y_1, \ldots, Y_n\}$ connects X-variables with Y-variables in one of the following ways ($=$ here denotes syntactical equality, with distinct we mean syntactical inequality):

1. $X_i\theta = Y_j\theta$ and for all $h = 1, \ldots, m$, $h \neq i$ and for all $k = 1, \ldots, n$, $k \neq j$, $X_h\theta$ and $Y_k\theta$ are all distinct from $X_i\theta$;

2. there are $j_1, \ldots, j_q$, $q \geq 2$, such that $X_i\theta = Y_{j_1}\theta = \cdots = Y_{j_q}\theta$ and for all $h = 1, \ldots, m$, $h \neq i$, and for all $k = 1, \ldots, n$, $k \neq j_1, \ldots, k \neq j_q$, $X_h\theta$ and $Y_k\theta$ are all distinct from $X_i\theta$; in this case we say there is a *q-fork* in the solution θ;

3. there are $i_1, \ldots, i_p$, $p \geq 2$, such that $X_{i_1}\theta = \cdots = X_{i_p}\theta = Y_j\theta$ and for all $h = 1, \ldots, m$, $h \neq i_1, \ldots, h \neq i_p$, and for all $k = 1, \ldots, n$, $k \neq j$, $X_h\theta$ and $Y_k\theta$ are all distinct from $X_{i_1}\theta$; in this case we say there is a *p-cone* in the solution θ.

We will describe below a procedure for counting all such solutions.

Given $n > 0$, a n-tuple $c \equiv [i_1, \ldots, i_n]$ is said to be a *configuration* for the set $\{Y_1, \ldots, Y_n\}$ if the non-negative integers $i_1, \ldots, i_n$ are such that $\sum_{j=1}^{n} i_j \cdot j = n$.

Let k ($1 \leq k \leq n$) be $\sum_{j=1}^{n} i_j$; the configuration c is a witness of any partition of $\{Y_1, \ldots, Y_n\}$ into k nonempty disjoint subsets such that there are exactly i_1 singleton subsets, i_2 doubleton subsets, and so on. Let $\mathcal{C}_n$ be the set of all the configurations for a fixed n; for instance,

```
configurations(N, Cn) :−
    setof(C, conf(N, 1, N, C), Cn).
conf(0, _, M, [ ]) :−
    !, M = 0.
conf(N, _, 0, C) :−
    !, zerolist(N, C).
conf(N, W, M, [A | C]) :−
    T is (M div W), in(A, T),
    M1 is M − W ∗ A,
    N1 is N − 1, W1 is W + 1,
    conf(N1, W1, M1, C).

in(T, T).
in(A, T) :−
    T > 0, T1 is T − 1,
    in(A, T1).
zerolist(0, [ ]) :− !.
zerolist(N, [0 | R]) :−
    M is N − 1,
    zerolist(M, R).
```

Figure 4.9: : The PROLOG program generating configurations.

when $n = 6$, the possible configurations are the following eleven

i_1	i_2	i_3	i_4	i_5	i_6
6	0	0	0	0	0
4	1	0	0	0	0
3	0	1	0	0	0
2	0	0	1	0	0
1	0	0	0	1	0
0	0	0	0	0	1

i_1	i_2	i_3	i_4	i_5	i_6
2	2	0	0	0	0
0	3	0	0	0	0
0	0	2	0	0	0
1	1	1	0	0	0
0	1	0	1	0	0

It is easy to write a PROLOG program which computes the set $\mathcal{C}_n$, for a given n (see figure 4.9).

Let $\|[i_1, \ldots, i_n]\|$ be the number of partitions of $\{Y_1, \ldots, Y_n\}$ of the form uniquely determined by $[i_1, \ldots, i_n]$; clearly,

$$\sum_{\substack{i_1+\cdots+i_n=k \\ 1\cdot i_1+\cdots+n\cdot i_n=n}} \|[i_1, \ldots, i_n]\| = \left\{ {n \atop k} \right\}.$$

The following simple example will clarify how to compute $\|[i_1, \ldots, i_n]\|$; let $n = 16$ and $c \equiv [5, 4, 1, 0, \ldots, 0]$.
There are $\binom{16}{2}$ ways for selecting the first doubleton, $\binom{14}{2}$, $\binom{12}{2}$, and $\binom{10}{2}$ ways for selecting the second, the third and the fourth doubleton,

respectively. In this way, however, any partition will be counted 4! (= i_2!) times, since there is no difference in selecting two elements Y_i and Y_j in the first, second, third or fourth attempt.
There are still $\binom{8}{3}$ for selecting the unique subset of three elements. The remaining elements will constitute the singletons of the partition. Thus,

$$\|c\| = \frac{\binom{16}{2}\binom{14}{2}\binom{12}{2}\binom{10}{2}}{4!} \cdot \frac{\binom{8}{3}}{1!} = \frac{16!}{((2!)^4(3!)^1)(5!4!1!)}\,.$$

Such situation is easy to generalize so as to obtain:

$$\|[i_1,\ldots,i_n]\| = \frac{n!}{\Pi_{j=2}^{n}(j!)^{i_j}\cdot\Pi_{j=1}^{n}(i_j!)}\,.$$

Assume a partition S of $\{Y_1,\ldots,Y_n\}$ ($S \subseteq \mathcal{P}(\{Y_1,\ldots,Y_n\})$) has the configuration $c \equiv [i_1,\ldots,i_n]$. We want to compute the number of (most general and independent) solutions θ to problem (2) such that S is the smallest set fulfilling $S = \{Z : (\forall Y_iY_j \in Z)(Y_i\theta = Y_j\theta)\}$.

Figure 4.10 illustrates the form of any such θ:

- subsets of $\{X_1,\ldots,X_m\}$ consisting of $i_2, i_3, \ldots, i_n$ elements are selected;
- for any $j = 2,\ldots,n$, θ connects with a bijection the subset identified by i_j to the subset of S constituted by sets of exactly j elements.
- Moreover, the remaining $m - \sum_{j=2}^{n} i_j$ elements of $\{X_1,\ldots,X_m\}$ are connected by a surjection with the remaining i_1 elements of $\{Y_1,\ldots,Y_n\}$.

Thus, there are

$$\binom{m}{i_2}\binom{m-i_2}{i_3}\cdots\binom{m-\sum_{j=2}^{n-1} i_j}{i_1}$$

ways for choosing the elements reflecting the situation described; once they are fixed, there are

$$\mathit{Surj}(i_2,i_2)\cdots\mathit{Surj}(i_n,i_n)\mathit{Surj}\Big(m-\sum_{j=2}^{n} i_j, i_1\Big)$$

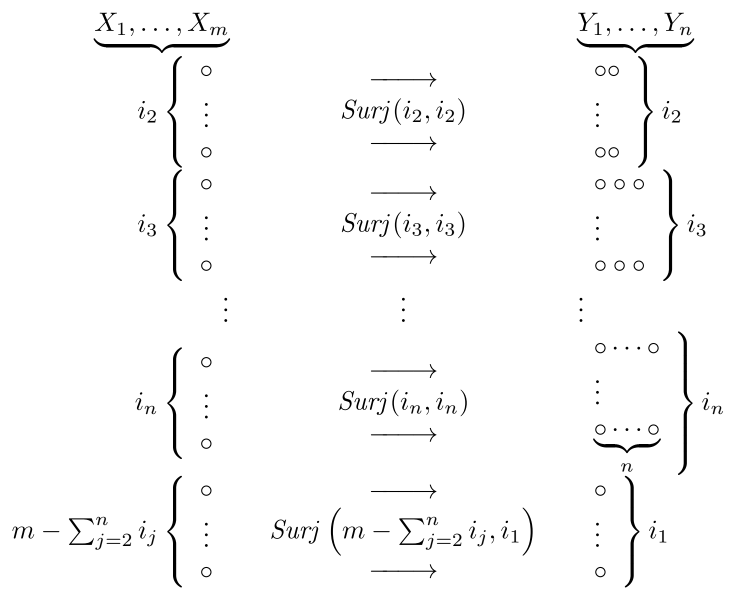


Figure 4.10: The form of a generic solution.

possible situations. Hence, the number of m.g.u.'s we are looking for is

$$\frac{m!}{\left(m - \sum_{j=2}^{n} i_j\right)!} \cdot Surj\left(m - \sum_{j=2}^{n} i_j, i_1\right) = \frac{m! \cdot i_1!}{(m - \sum_{j=2}^{n} i_j)!} \cdot \left\{ {m - \sum_{j=2}^{n} i_j \atop i_1} \right\}.$$

Finally, given a configuration $c \equiv [i_1, \dots, i_n]$, the number of possible m.g.u.'s from the set $\{X_1, \dots, X_m\}$ to any partition of $\{Y_1, \dots, Y_n\}$ having configuration c (we call this number $\|c\|^m$) will be

$$\begin{aligned} \|[i_1, \dots, i_n]\|^m &= \|[i_1, \dots, i_n]\| \cdot \frac{m! \cdot i_1!}{\left(m - \sum_{j=2}^{n} i_j\right)!} \cdot \left\{ {m - \sum_{j=2}^{n} i_j \atop i_1} \right\} \\ &= \frac{m! \cdot n! \cdot i_1!}{\left(\Pi_{j=2}^{n} (j!)^{i_j}\right) \cdot \left(\Pi_{j=1}^{n} (i_j!)\right) \cdot \left(m - \sum_{j=2}^{n} i_j\right)!} \cdot \left\{ {m - \sum_{j=2}^{n} i_j \atop i_1} \right\}. \end{aligned}$$

With this alternative point of view, the function Φ can be defined as $\Phi(m, n) = \sum_{c \in \mathcal{C}_n} \|c\|^m$.

In [103] an interesting approach in finding the solution to problem (2) using Taylor's series, is presented. In particular, it is implicitly proved that $\Phi(m, n) = (\Delta_x^m \Delta_y^n e^{(x(e^y - 1) + y(e^x - 1) - xy)})_{\langle 0,0 \rangle}$, where $\Delta_v^k f(\cdots, v, \cdots)$ means to derive k times with respect to the variable v.

The problem (3)

Any solution θ to problem

$$(3) \qquad \{X_1, \ldots, X_m, a_1, \ldots, a_k\} \;\doteq\; \{Y_1, \ldots, Y_n, a_1, \ldots, a_k\},$$

will map every element of $\{X_1, \ldots, X_m, a_1, \ldots, a_k\}$ into an element of $\{Y_1, \ldots, Y_n, a_1, \ldots, a_k\}$. Clearly, for $i = 1, \ldots, k$, a_i is implicitly mapped into itself.

Assume θ has the form $[X_{i_1}/a_\ell, \ldots, X_{i_\alpha}/a_\ell, Y_{j_1}/a_\ell, \ldots, Y_{j_\beta}/a_\ell] \cup \theta'$, where $\alpha, \beta > 1$. Such θ is an instance of the substitution $\theta'' \cup \theta'$, for any θ'' solution to the problem of type (2) $\{X_{i_1}, \ldots, X_{i_\alpha}\} \doteq \{Y_{j_1}, \ldots, Y_{j_\beta}\}$.

This means, in particular, that we do not have to count solutions in which both X_i/a_ℓ and Y_j/a_ℓ occur in the solution, for any i, j, and ℓ.

Any most general solution θ to problem (3) can be obtained as follows:

- choose two disjoint subsets S_0 and S_1 of $\{a_1, \ldots, a_k\}$;
- choose
  - a subset T_0 of $\{X_1, \ldots, X_m\}$ and
  - a subset T_1 of $\{Y_1, \ldots, Y_n\}$;
- θ is $\theta_0 \cup \theta_1 \cup \theta_2$;
- θ_i is a solution to the problem of type (1) $T_i \doteq S_i$, for $i = 0, 1$;
- θ_2 is a solution to the problem of type (2) $\{X_1, \ldots, X_m\} \setminus T_0 \doteq \{Y_1, \ldots, Y_n\} \setminus T_1$.

Hence, the number of most general and independent solutions to problem (3) is:

$$\begin{aligned}\Psi(m,n,k) \;=\; & \sum_{i=0}^{k} \binom{k}{i} \sum_{j=0}^{k-i} \binom{k-i}{j} \sum_{a=0}^{m} \binom{m}{a} \sum_{b=0}^{n} \binom{n}{b} \\ & \Big(Surj(a,i) \cdot Surj(b,j) \cdot \Phi(m-a, n-b) \Big).\end{aligned}$$

The problem (4)

From any solution θ to problem (3) a corresponding solution to problem

$$(4) \qquad \{X_1, \ldots, X_m, Z_1, \ldots, Z_k\} \;\doteq\; \{Y_1, \ldots, Y_n, Z_1, \ldots, Z_k\}\,,$$

can be obtained replacing any occurrence of a_i with Z_i, for $i = 1, \ldots, k$.

Problem (4) has a number of solutions strictly greater than problem (3), since it is consistent to consider solutions containing *p-forks*, $p \leq k$, proceeding of the unification problems $T_0 \doteq S_0$ and $T_1 \doteq S_1$. However any solution $\theta = [Z_{i_1}/X_i, \ldots, Z_{i_p}/X_i] \cup \theta'$, $p \leq k$ of the form described above is an instance of any of the solutions $\theta_{i_1} = [Z_{i_1}/X_i] \cup \theta'$, …,$\theta_{i_p} = [Z_{i_p}/X_i] \cup \theta'$.

Thus, problem (4) has $\underline{\text{exactly}}$ the same number of most general and independent solutions as problem (3).

The problem (5)

Any solution to the problem

$$(5) \qquad \{X_1, \ldots, X_m \,|\, Z\} \;\doteq\; \{a_1, \ldots, a_n\}\,,$$

can be computed as follows: let S be a nonempty subset of $\{a_1, \ldots, a_n\}$ and let θ_1 be a solution to the problem of type (1) $\{X_1, \ldots, X_m\} \doteq S$; then any substitution θ extending θ_1 with the mapping $Z/(\{a_1, \ldots, a_n\} \setminus S) \cup T$, for any $T \subseteq S$, is a solution to (5). It is easy to see that all the solutions obtained in this way are independent and, furthermore, the collection of them is a complete set of unifiers. The total number of solutions is therefore

$$\textstyle\sum_{i=1}^{n} \binom{n}{i} \Big(Surj(m, i) \cdot 2^i \Big)\,.$$

The problem (6)

Let S be a nonempty subset of $\{Y_1, \ldots, Y_n\}$ and let θ_1 be a solution to the problem of type (2) $\{X_1, \ldots, X_m\} \doteq S$; then any substitution θ extending θ_1 with the mapping

$$Z \;=\; (\{Y_1, \ldots, Y_n\} \setminus S) \cup T$$

is a solution to problem

$$(6) \qquad \{X_1, \ldots, X_m \,|\, Z\} \;\doteq\; \{Y_1, \ldots, Y_n\}\,,$$

for any $T \subseteq S$. However, in this case, solutions are not all pairwise independent, as it is shown by the following example: consider the problem $\{X_1 \,|\, Z\} \doteq \{Y_1, Y_2\}$. Let $S = \{Y_1, Y_2\}$, then $\{X_1\} \doteq \{Y_1, Y_2\}$ has the unique solution $[Y_1/X_1, Y_2/X_1]$. Such a solution can be extended with $[Z/\{Y_2\}]$. Let now $S = \{Y_1\}$, then the problem $\{X_1\} \doteq \{Y_1\}$ has the unique solution $[X_1/Y_1]$. Such a solution can be extended with $[Z/\{Y_2\}]$, a more general solution than the first presented.

A closer analysis of the solutions to the problem $\{X_1, \dots, X_m\} \doteq S$ must be performed in order to identify whether a solution is general or not.

Given a solution θ to $\{X_1, \dots, X_m\} \doteq S$, we want to extend it with a substitution for Z. Such substitution should be of the form $[Z/(\{Y_1, \dots, Y_n\} \setminus S) \cup T]$, with $T \subseteq S$.

Assume an element Y_i of S belongs to T. Two cases are possible:

1. θ does not connect Y_i with any Y_j, for $j = 1, \dots, n$, $i \neq j$. In other words

$$\begin{array}{ll} X_{i_1}\theta = \cdots = X_{i_k}\theta = Y_i\theta\,, & \text{for some } i_1, \dots, i_k \in \{1, \dots, m\},\ k \geq 1, \\ Y_i\theta \neq Y_j\theta & \text{for } j = 1, \dots, n,\ i \neq j. \end{array}$$

2. θ connects Y_i with some Y_j's, for $j = 1, \dots, n$, $i \neq j$. This means (see problem (2)) that there is a k-fork in the solution θ:

$$\begin{array}{l} X_h\theta = Y_i\theta = Y_{i_1}\theta = \cdots = Y_{i_k}\theta \\ \text{for exactly one } h \in \{1, \dots, m\} \text{ and} \\ \text{for some } i_1, \dots, i_k \in \{1, \dots, n\}. \end{array}$$

In the former case we can insert Y_i into T. In fact, if we consider the sub-problem obtained by removing it from S, there are no possibilities to map $X_{i_1}, \dots, X_{i_k}$ in a way that subsumes such solution.

In the latter case the situation is radically different. Consider the the sub-problem obtained by removing Y_i from S, we get the solution θ' equal to θ except that Y_i is not considered (i.e. $X_h\theta' = Y_{i_1}\theta' = \cdots = Y_{i_k}\theta'$). This means that if we inserted Y_i into T, we would generate an instance of a solution already computed. An analogous reasoning can be performed for any of $Y_{i_1}, \dots, Y_{i_k}$ and for all k-forks in θ.

Thus, the number of such solutions can be computed using the concept of *configuration* defined to describe solutions to problem (2). Given a solution θ for the unification problem $\{X_1, \ldots, X_m\} \doteq S$, for some $S \subseteq \{Y_1, \ldots, Y_n\}$ such that $|S| = j$, consider its configuration $c_\theta = [i_1, \ldots, i_j]$. There are $2^{j - \mathit{forks}(\theta)}$ possible values for T, where $\mathit{forks}(\theta) = 2 \cdot i_2 + \cdots + j \cdot i_j$. Hence,

$$\sum_{j=1}^{n} \binom{n}{j} \sum_{\substack{\theta \text{ is a solution to} \\ \{X_1, \ldots, X_m\} \doteq \{Y_1, \ldots, Y_j\}}} 2^{j - \mathit{forks}(\theta)}\ .$$

The problem (7)

Let S_0 be a subset of $\{X_1, \ldots, X_m\}$ and S_1 be a subset of $\{Y_1, \ldots, Y_n\}$. Problem

$$(7) \qquad \{X_1, \ldots, X_m \,|\, Z\} \ \doteq\ \{Y_1, \ldots, Y_n \,|\, Z\}\,,$$

can be reduced to the family of problems

$$\begin{array}{rcl} S_0 & \doteq & S_1, \\ Z & \supseteq & (\{X_1, \ldots, X_m\} \setminus S_0) \cup (\{Y_1, \ldots, Y_n\} \setminus S_1) \end{array}$$

If θ is a solution to $S_0 \doteq S_1$, then

$$\theta \cup [Z/(\{X_1, \ldots, X_m\} \setminus S_1) \cup (\{Y_1, \ldots, Y_n\} \setminus S_2) \cup N]\,,$$

where N is a new variable, whose intended meaning is 'any set', is a solution to problem (7). Furthermore, they are all pairwise independent. The number of most general and independent solutions to problem (7) is $\sum_{i=0}^{m} \binom{m}{i} \sum_{j=0}^{n} \binom{n}{j} \Phi(i, j)$.

The problem (8)

Problem

$$(8) \qquad \{X_1, \ldots, X_m \,|\, W\} \ \doteq\ \{Y_1, \ldots, Y_n \,|\, Z\}$$

can be reduced to the family of problems

$$\begin{array}{rcl} S_0 & \doteq & S_1, \\ Z & \doteq & (\{X_1, \ldots, X_m\} \setminus S_0) \cup N \cup T_0, \\ W & \doteq & (\{Y_1, \ldots, Y_n\} \setminus S_1) \cup N \cup T_1, \end{array}$$

where

- $S_0 \subseteq \{X_1, \dots, X_m\}$ and $S_1 \subseteq \{Y_1, \dots, Y_n\}$, and
- T_i is a subset of S_i, for $i = 0, 1$, and
- N is a new variable (whose intended meaning is 'any set').

Similarly to problem (6), we need to bound the range of the T_is in order to avoid the generation of instances of other generated solutions.

As shown in the analysis of the problem (2), any solution θ for the problem $S_0 \doteq S_1$ (instance of the problem (2)) can give rise to three situations.

X_i and Y_j of case 1 can be inserted into T_0 and T_1, respectively, but not simultaneously. In fact the solution

$$\{X_i = Y_j, \dots, Z = \{\cdots \mid N\}, W = \{\cdots \mid N\}\}$$

is more general than

$$\{X_i = Y_j, \dots, Z = \{\cdots, X_i \mid N\}, W = \{\cdots, Y_j \mid N\}\}\,.$$

Following the identical reasoning to the one performed in the analysis of problem (6), X_i of case 2 can be inserted into T_0, while the introduction of Y_{j_ℓ}, for $\ell = 1, \dots, k$ in T_1 generates an instance of another solution.

Similarly, Y_j of case 3 can be inserted into T_1, while X_{i_ℓ}, for $\ell = 1, \dots, k$ must not be introduced in T_0, if we want to guarantee the minimality property of the solutions.

To sum up, given a solution θ to $S_0 \doteq S_1$, we define as $vert_0(\theta)$ the sum of number of k-forks for $k = 2, \dots, |S_1|$, and $vert_1(\theta)$ the sum of the number of h-forks for $h = 2, \dots, |S_0|$. $cones(\theta)$ is defined to be $\sum_{h=0}^{|S_0|} h \cdot (\#\text{ of } h\text{-cones})$, and $forks(\theta)$ is the same function defined in the solution to problem (6).

Clearly, $|S_0| - cones(\theta) - vert_0(\theta) = |S_1| - forks(\theta) - vert_1(\theta)$; such number (say p) is the number of elements connected with a bijection (see case 1 above). As it has already been explained, such elements can be inserted in T_0 and T_1 not simultaneously: there are $\mid \{\langle A, B\rangle : A, B \subseteq \{1, \dots, p\}, A \cap B = \emptyset\} \mid = 3^p$ possibilities to extend θ.

```
naive(A, B) :-
    (var(A); var(B)), !, A = B.
naive(A, B) :-
    A =.. [F | Alist],
    B =.. [F | Blist],
    F ≠ '.', !,
    naive_all(Alist, Blist).
naive([T | Trest], [S | Srest]) :-            (i)
    naive(T, S), naive(Trest, Srest).
naive([T | Trest], [S | Srest]) :-            (ii)
    naive(T, S), naive([T | Trest], Srest).
naive([T | Trest], [S | Srest]) :-            (iii)
    naive(T, S), naive(Trest, [S | Srest]).
naive([T | Trest], [S | Srest]) :-            (iv)
    naive([T | New], Srest),
    naive(Trest, [S | New]).
```

Figure 4.11: The PROLOG code for a naive set unification algorithm

Thus, we are ready to count all the solutions to the problem (8):

$$\sum_{a=0}^{m} \binom{m}{a} \sum_{b=0}^{n} \binom{n}{b} \sum_{\substack{\theta \text{ is a solution to} \\ \{X_1,\dots,X_a\} = \{Y_1,\dots,Y_b\}}} 2^{vert_0(\theta)} 2^{vert_1(\theta)} 3^{a-vert_0(\theta)-cones(\theta)} .$$

Note that if we chose T_0 and T_1 as any subsets of S_0 and S_1, respectively, the total number of computed unifiers would be

$$\Delta(m,n) = \sum_{i=0}^{m} \binom{m}{i} \sum_{j=0}^{n} \binom{n}{j} \Big(2^{i+j} \cdot \Phi(i,j)\Big).$$

4.4.2 Solutions computed by a naive algorithm

The PROLOG code of Fig. 4.11 is the core of the general set unification algorithm presented in [55]; as that algorithm, it does not terminate for problem (7) (same rest variables). The algorithm presented in § 4.2.4 extends it covering also this case. Functional symbols $\emptyset$ and $\{\cdot \mid \cdot\}$ are represented by $[\,]$ and $[\cdot \mid \cdot]$, respectively.

Predicate naive_all is recursively defined on lists in the obvious way. naive algorithm has a minimal behavior for problem (1) only. This is stated in the following Theorems.

Problem (1)

Lemma 4.51 *The* PROLOG *execution of the goal*

:– naive($[X_1 \mid N], [a_1, \ldots, a_n]$)

generates $2n$ solutions. More precisely, for $i = 1, \ldots, n$, they are of the form

$X_1 = a_i, N = [a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n]$ *, and*
$X_1 = a_i, N = [a_1, \ldots, a_n]$ *.*

Proof. It follows immediately from an analysis of the SLD tree.

4.51 □

Theorem 4.52 *Let $f_1(m, n)$ be the number of solutions computed by the* PROLOG *execution of the goal*

:– naive($[X_1, \ldots, X_m], [a_1, \ldots, a_n]$).

Then

$$\begin{cases} f_1(m,n) &= 0 & \textit{if } m < n \\ f_1(m,1) &= 1 & \textit{if } m > 0 \\ f_1(m,n) &= n(f_1(m-1,n-1) + f_1(m-1,n)) & \textit{if } m \geq n > 1\,. \end{cases}$$

Proof. The non-deterministic choice (*ii*) leads always to a failure situation, due to the fact that $a_1, \ldots, a_n$ are distinct constants. For the same reasons, if $m < n$, there are no solutions.
If $n = 1$ the result follows immediately by a simple analysis of the SLD-tree.
Assume $1 < n \leq m$. Then the non-deterministic choice

(*i*) reduces to the problem of size $(m-1, n-1)$:
:– naive($[X_2, \ldots, X_m], [a_2, \ldots, a_n]$).

(*iii*) reduces to the problem of size $(m-1, n)$:
:– naive($[X_2, \ldots, X_m], [a_1, \ldots, a_n]$).

(*iv*) The goal $:\!-\, \mathsf{naive}([X_1 \mid N],[a_2,\dots,a_n])$ is computed. Lemma 4.51 ensures that it generates
$n-1$ goals of the form
$:\!-\, \mathsf{naive}([X_2,\dots,X_m],[a_1,\dots,a_n])$, and
$n-1$ goals of the form
$:\!-\, \mathsf{naive}([X_2,\dots,X_m],[a_1,\dots,a_{i-1},a_{i+1},\dots,a_n])$.

4.52 □

Observe that the function f of Theorem 4.52 is exactly the function *Surj*. Hence,

Corollary 4.53 *The naive set unification algorithm is minimal with respect to sample problem* (1)*.*

Problem (2)

We compute the function $f_2 : \omega^2 \to \omega$ which returns, for any m and n, the number of computed solutions by the goal

$$:\!-\, \mathsf{naive}([X_1,\dots,X_m],[Y_1,\dots,Y_n])\,.$$

It is easy to see that $f_2(m,1) = 1$ and $f_2(1,n) = 1$ for any $m,n \geq 1$. For the general case, the following preliminary results are useful.

Lemma 4.54 *The number of solutions generated by the* Prolog *execution of the goal*

$:\!-\, \mathsf{naive}([X_1 \mid N],[Y_1,\dots,Y_n])$

is $\sum_{i=1}^{n} 2^i = 2^{n+1}-2$*. Furthermore, let* $\#(n,k)$ *be the number of solutions generated by the execution of the goal above such that* N *is mapped to a list of* k *elements* ($0 \leq k \leq n$)*. Then*

$$\begin{cases} \#(n,i) &=& \binom{n+1}{i} \quad \textit{for } i = 0,\dots,n-1, \\ \#(n,n) &=& n\,. \end{cases}$$

Proof. Let $g(n)$ be the function which returns the number of solutions generated by the Prolog execution of the goal

$$:\!-\, \mathsf{naive}([X_1,\dots,X_m],[Y_1,\dots,Y_n])\,.$$

It is easy to see that $g(1) = 2$, and that $g(n+1) = 2 + 2g(n)$.
By induction on $n \geq 1$ we prove that $g(n) = \sum_{i=1}^{n} 2^i$.
The base case is trivial. For the induction step, $g(n+1) = 2+2g(n) = 2 + 2\sum_{i=0}^{n} 2^i = \sum_{i=0}^{n+1} 2^i$.
This ensures that, for any n,

$$\sum_{i=0}^{n} \#(n,i) = 2^{n+1} - 2 \ .$$

Analyzing the SLD-tree for the goal, it is easy to see that $\#(n,n) = n$.
Hence,

$$\begin{array}{l} \sum_{i=0}^{n-1} \#(n,i) + n = 2^{n+1} - 2, \text{ i.e.} \\ \sum_{i=0}^{n-1} \#(n,i) + (n+1) + 1 = 2^{n+1} \ . \end{array}$$

Since that property holds for any n, even without entering into the details of the SLD tree, it is not difficult to become convinced that

$$\#(n,i) \quad = \quad \binom{n+1}{i} \quad \text{for } i = 0, \ldots, n-1.$$

4.54 □

We are ready for the final result

Theorem 4.55 *Let $f_2(m,n)$ be the number of solutions computed by the* Prolog *execution of the goal*

$$:\!-\ \mathsf{naive}([X_1, \ldots, X_m], [Y_1, \ldots, Y_n]).$$

Then

$$\begin{cases} f_2(m,1) & = & 1 \\ f_2(1,n) & = & 1 \\ f_2(m,n) & = & f_2(m-1,n-1) + f_2(m,n-1) + \qquad m,n > 1 \\ & & nf_2(m-1,n) + \sum_{i=0}^{n-2} \binom{n}{i} f_2(m-1,i+1) \ . \end{cases}$$

Proof. Base cases follow trivially. Assume $m, n > 1$. $f_2(m,n)$ is the sum of

- $f_2(m-1,n-1)$: situation (i);
- $f_2(m,n-1)$: situation (ii);
- $f_2(m-1,n)$: situation (iii);

- in the situation (iv) the solutions to $:-\mathsf{naive}([X_1 \mid N], [Y_2, \ldots, Y_n])$ should be computed (Lemma 4.54 ensures they are $2^n - 2$). Any of them comprises a substitution of the form $[N/[Y_{i_1}, \ldots, Y_{i_k}]]$.

  Then the goal $:-\ \mathsf{naive}([X_2, \ldots, X_m], [Y_1, Y_{i_1}, \ldots, Y_{i_k}])$ is computed.

  From Lemma 4.54 we have to sum:

  $\sum_{i=0}^{n-1} \#(n-1, i) f_2(m-1, i+1) =$

  $\sum_{i=0}^{n-2} \binom{n}{i} f_2(m-1, i+1) + (n-1) f_2(m-1, n)$.

Summing the four addends, the result follows.

4.55 □

Problem (4)

The naive program treats problems (2) and (4) exactly at the same way. This means that the number of solutions computed by the Prolog execution of the goal

$$:-\ \mathsf{naive}([X_1, \ldots, X_m, Z_1, \ldots, Z_k], [Y_1, \ldots, Y_n, Z_1, \ldots, Z_k]),$$

where $X_1, \ldots, X_m, Y_1, \ldots, Y_n, Z_1, \ldots, Z_k$ are pairwise distinct variables, is $f_2(m+k, n+k)$. For instance, when $m = 2$, $n = 2$, and $k = 3$, 95401 solutions are computed instead of the 56 needed.

Problem (7)

As already sketched, presented naive program is not sufficient to deal with problem (7) (same rest variables). However, referring to the Prolog implementation of the complete algorithm presented in 4.2.4 (assume it is named naive*), it is easy to prove that

Lemma 4.56 *The function f_7 returning the number of solutions computed by the* Prolog *execution of the goal*

$$:-\ \mathsf{naive}^*([X_1, \ldots, X_m \mid Z], [Y_1, \ldots, Y_n \mid Z]),$$

where $X_1, \ldots, X_m, Y_1, \ldots, Y_n, Z$ are pairwise distinct variables, is recursively defined as follows:

$$\begin{cases} f_7(0,n) &= 1 & n \geq 0 \\ f_7(m,0) &= 1 & m > 0 \\ f_7(m,n) &= n \cdot (f_7(m-1,n-1) + f_7(m,n-1)) + & m > 0, n > 0 \\ & \quad (n+1) f_7(m-1,n)\,. & \end{cases}$$

Problem (8)

We describe the function $f_8 : \omega^2 \to \omega$ which returns the number of answers computed by the PROLOG execution of the goal

$:- \mathsf{naive}([X_1, \ldots, X_m \,|\, W], [Y_1, \ldots, Y_n \,|\, Z])$

where $X_1, \ldots, X_m, Y_1, \ldots, Y_n, W, Z$ are pairwise distinct variables.

First the following two technical Lemmata are needed:

Lemma 4.57 *The* PROLOG *execution of the goal*

$:- \mathsf{naive}([X_1 \,|\, W], [Y_1, \ldots, Y_n \,|\, Z])$

where $X_1, \ldots, X_m, Y_1, \ldots, Y_n, W, Z$ are pairwise distinct variables returns $3 \cdot 2^n - 2$ solutions.

Proof. Call $g : \omega \to \omega$ the function returning such number.
It is easy to see that $g(1) = 4$: the solutions returned to the goal

$:- \mathsf{naive}([X_1 \,|\, W], [Y_1 \,|\, Z])$

are the following:

- using clause (i): $X_1 = Y_1, W = Z$;
- using clause (ii): $X_1 = Y_1, W = [Y_1 \,|\, Z]$;
- using clause (iii): $X_1 = Y_1, [X_1 \,|\, W] = Z$;
- using clause (iv): $W = [Y_1 \,|\, N], Z = [X_1 \,|\, N]$.

Analogously, $g(n) = 2 + 2g(n-1)$, when $n > 1$: the goal

$:- \mathsf{naive}([X_1 \,|\, W], [Y_1, \ldots, Y_n \,|\, Z])$

is handled as follows:

- using clause (i) the solution $X_1 = Y_1, W = [Y_2, \dots, Y_n \,|\, Z]$ is returned;
- using clause (iii) the solution $X_1 = Y_1, W = [Y_1, \dots, Y_n \,|\, Z]$ is returned;
- using clause (ii) the substitution $X_1 = Y_1$ is obtained, and the goal
  :– $\mathsf{naive}([X_1 \,|\, W], [Y_2, \dots, Y_n \,|\, Z])$,
  of size $n - 1$, is computed;
- using clause (iv) the substitution $W = [Y_1 \,|\, N]$ is obtained, and the goal
  :– $\mathsf{naive}([X_1 \,|\, N], [Y_2, \dots, Y_n \,|\, Z])$,
  of size $n - 1$, is computed.

It is easy to prove by induction on n, that the unique solution to $g(n)$ is $3 \cdot 2^n - 2$.

4.57 □

Lemma 4.58 *Let $\#(n, k)$ be the number of solutions generated by the* PROLOG *execution of the goal*

:– $\mathsf{naive}([X_1 \,|\, W], [Y_1, \dots, Y_n \,|\, Z])$

such that W is mapped to a list of k elements ($0 \leq k \leq n$). Then

$$\begin{cases} \#(n,i) &= \binom{n+1}{i} + \binom{n}{i} \quad \textit{for } i = 0, \dots, n-1, \\ \#(n,n) &= n + 1\,. \end{cases}$$

Proof. It is easy to see (see proof of Lemma 4.57) that the recursive definition of $\#(n, k)$ is the following:

$$\begin{cases} \#(m,n) &= 0 & \text{if } m < n \text{ or } m = 0 \\ \#(n,0) &= 2 & \text{if } 0 < n \\ \#(n,i) &= \#(n-1,i) + \#(n-1,i-1) & \text{if } 0 < i < n-1 \\ \#(n,n-1) &= 1 + \#(n-1,n-1) + \#(n-1,n-2) & \\ \#(n,n) &= 1 + \#(n-1,n-1)\,. & \end{cases}$$

It is straightforward to prove $\#(n, n) = n + 1$ and, consecutively, $\#(n, n-1) = n + \frac{(n+1)n}{2}$, and so on. However, from Lemma 4.57, we know that $\sum_{i=0}^{n} \#(n, i) = 3 \cdot 2^n - 2$. Observe that

$$\begin{array}{lcll}
3 \cdot 2^n - 2 & = & 2 \cdot 2^n + 2^n - 2 & = \\
2^{n+1} + 2^n - 2 & = & \sum_{i=0}^{n+1} \binom{n+1}{i} + \sum_{i=0}^{n} \binom{n}{i} - 2 & = \\
\sum_{i=0}^{n} \left(\binom{n+1}{i} + \binom{n}{i} \right) - 1 \, . & & &
\end{array}$$

It is a matter of routine to complete the proof.

4.58 □

Theorem 4.59 *The function f_8 described above can be defined as follows:*

$$\begin{cases}
f_8(1,1) = 4 & \\
f_8(m,1) = 3 \cdot 2^m - 2 & \textit{if } m > 1 \\
f_8(1,n) = 3 \cdot 2^n - 2 & \textit{if } n > 1 \\
f_8(m,n) = f_8(m-1,n-1) + f_8(m,n-1) + & \textit{if } m,n > 1 \\
\qquad (n+1) f_8(m-1,n) + & \\
\qquad \sum_{i=0}^{n-2} \left(\binom{n}{i} + \binom{n-1}{i} \right) f_8(m-1,i+1) \, . &
\end{cases}$$

Proof. The proof for $f_8(1,1)$, $f_8(m,1)$, and $f_8(1,n)$ is the same as in Lemma 4.57.
When m and n are both greater than 1, the goal

$:\!-\ \mathsf{naive}([X_1, \ldots, X_m \,|\, W], [Y_1, \ldots, Y_n \,|\, Z])$

is handled as follows:

- using clause (i) the substitution $X_1 = Y_1$ is returned, and the goal
  $:\!-\ \mathsf{naive}([X_2, \ldots, X_m \,|\, W], [Y_2, \ldots, Y_n \,|\, Z])$,
  of size $(m-1, n-1)$ is computed;

- using clause (ii) the substitution $X_1 = Y_1$ is returned, and the goal
  $:\!-\ \mathsf{naive}([X_1, \ldots, X_m \,|\, W], [Y_2, \ldots, Y_n \,|\, Z])$,
  of size $(m, n-1)$ is computed;

- using clause (iii) the substitution $X_1 = Y_1$ is returned, and the goal
  $:- \mathsf{naive}([X_2, \ldots, X_m \mid W], [Y_1, \ldots, Y_n \mid Z]),$
  of size $(m-1, n)$ is computed;

- using clause (iv) first the goal
  $:- \mathsf{naive}([X_1 \mid N], [Y_2, \ldots, Y_n \mid Z])$
  is computed, reporting solutions θ's for N of the form described in Lemma 4.57. After that, the goal
  $:- \mathsf{naive}([X_2, \ldots, X_m \mid W], [Y_1 \mid N\theta])$
  is handled. In particular, n sub-problems of size $(m-1, n)$ and $\#(n-1, i)$ (see Lemma 4.58) sub-problems of size $(m-1, \#(n-1, i)+1)$ are computed.

From that description, the above recursive definition for f is uniquely determined.

4.59 □

4.4.3 The algorithm SUA

We now introduce the algorithm SUA (Set Unification Algorithm) which has been proved to be minimal in [11, 10] for the eight sample problems presented. Any non-deterministic computation of SUA can be seen as a particular non-deterministic computation of the execution of algorithm Unify_set presented in § 4.2.4. In particular, actions 1–8 of SUA are exactly the same as the ones in Unify_set. This ensures its correctness and its termination, and, moreover, the completeness of actions 1–8. The completeness of the set-to-set actions 9–13 is ensured by the fact that it returns the required number of solution to the general problems (1), (6), (7), and (8), which constitute the four templates into which any set unification problem is comprised (see remark 4.50).

In order to make the description of the algorithm as clear as possible, some local notation will be defined.

The *set-operations* '$|\cdot|$' (cardinality), '$\subseteq$' (inclusion), '$\subset$' (strict inclusion), '$\cup$', (union) '$\cap$', (intersection) and '$-$' (set difference) will be used on terms denoting sets. The meaning of the set operators is purely syntactical; for instance $|\{X_1, X_2\}| = 2$: we do not need to distinguish the two cases $X_1 = X_2 \wedge |\{X_1, X_2\}| = 1$ and $X_1 \neq X_2 \wedge |\{X_1, X_2\}| = 2$. The function *no_dup* which removes duplicates in a term representing a set ensures no ambiguity in using them.

SUA takes as input a system of equations $\mathcal{E}$ between terms and returns either *fail*—$\mathcal{E}$ is not unifiable—or, non-deterministically, a substitution θ. The set of all such θ's constitutes a complete set of T-unifiers. In the algorithm, X, W, Z and Z' denote generic variables, $t, t_1, t_2, \ldots, s_1, s_2, \ldots$ denote generic terms, N denotes a new variable introduced by SUA. k and k' will denote non-variable terms which main functional symbol distinct from $\{\cdot \mid \cdot\}$.

$FV(\ell)$ represents the set of variables occurring in the term ℓ, while $\theta|_S$ constraints the domain of θ to the variables contained in S. n.d. is a denotation for non-deterministically. The algorithm temporarily generates equations marked by '$*$'; they are called *active* equations and they are immediately removed by action 0. The algorithm SUA is presented in Fig. 4.12.

Active equations $s \overset{*}{=} t$ should be handled before the others in order to guarantee termination. They are temporarily introduced by actions 4, 9, 11, 12, and 13. This is sufficient to simulate the termination strategy imposed by the data structure stack in the algorithm Unify_set.

The function unify_set takes as input two terms S_1 and S_2 representing closed and nonempty sets; it selects non-deterministically which equalities between elements of S_1 and S_2 should accompany the system $\mathcal{E}$ in a recursive call to SUA.

```
function unify_set($S_1, S_2$);
Let $\{t_1, \ldots, t_m\} = no\_dup(S_1)\,; \{s_1, \ldots, s_n\} = no\_dup(S_2)\,;$
1. If $\{t_1, \ldots, t_m\}$ and $\{s_1, \ldots, s_n\}$ are syntactically equal then return SUA($\mathcal{E}$)
2. elseif $m = 1$ and $n > 1$ then return $\{s_i \doteq t_1 : 1 \leq i \leq n\}$;
3. elseif $m \geq 1$ and $n = 1$ then return $\{t_i \doteq s_1 : 1 \leq i \leq m\}$
4. else $Common\_part$ := $\{t_1, \ldots, t_m\} \cap \{s_1, \ldots, s_n\}\,;$
            $Disagr_1$ := $\{t_1, \ldots, t_m\} - Common\_part\,;$
            $Disagr_2$ := $\{s_1, \ldots, s_n\} - Common\_part\,;$
  (a) if $Common\_part = \emptyset$
      then fix an $i \in \{1, \ldots, m\}$: select n.d. one of the following actions:
        i. return $\overset{1}{=} (t_i, \{t_1, \ldots, t_m\}, \{s_1, \ldots, s_n\})$;
```

```
SUA($\mathcal{E}$);
If $\mathcal{E}$ is in solved form then return $\mathcal{E}$
elseif $\mathcal{E}^*$ is not empty
then choose n.d. one $active$ equation $s \overset{*}{=} t$ in $\mathcal{E}^*$; $\mathcal{E}' := \mathcal{E} - \{s \overset{*}{=} t\}$;
0.  i. $s \equiv X$: if $X$ occurs in $t$ then $fail$ else return SUA($\mathcal{E}'[X/t] \cup \{X \doteq t\}$);
    ii. $s \equiv f(s_1, \dots, s_m)$ and $t \equiv f'(t_1, \dots, t_n)$: if $f \not\equiv f'$ then $fail$ else
        (i.e. $f \equiv f'$ and $m = n$): return SUA($\{s_1 \doteq t_1, \dots, s_n \doteq t_n\} \cup \mathcal{E}'$);
else choose n.d. one equation $e$ in $\mathcal{E}$; $\mathcal{E}' := \mathcal{E} - \{e\}$; case $e$ of:
1.  $X \doteq X$: return SUA($\mathcal{E}'$);
2.  $t \doteq X$ and $t$ is not a variable: return SUA($\{X \doteq t\} \cup \mathcal{E}'$);
3.  $X \doteq t$, $X$ is a variable not occurring in $t$: return SUA($\mathcal{E}'[X/t] \cup \{X \doteq t\}$);
4.  $X \doteq \{t_1, \dots, t_m \,|\, X\}$: return SUA($\mathcal{E} \cup \{X \overset{*}{=} \{t_1, \dots, t_m \,|\, N\}\}$);
5.  $X \doteq \{t_1, \dots, t_m \,|\, t\}$, where $t$ is a variable or $t \equiv f(t_1, \dots, t_n)$, $f \not\equiv \{\cdot \,|\, \cdot\}$,
    and $X$ occurs in $t_0$, or ..., or in $t_m$, or $t$ is not a variable and
    $X$ occurs in $t$: $fail$;
6.  $X \doteq t$, and $t \equiv f(t_1, \dots, t_n)$, $f \not\equiv \{\cdot \,|\, \cdot\}$ and
    $X$ is a variable occurring in $t$: $fail$;
7.  $f(s_1, \dots, s_n) \doteq g(t_1, \dots t_m)$, where $f, g \in \Sigma$, $f \not\equiv g$: $fail$;
8.  $f(s_1, \dots, s_n) \doteq f(t_1, \dots t_n)$, $f \in \Sigma$, $f \not\equiv \{\cdot \,|\, \cdot\}$:
    return SUA($\mathcal{E}' \cup \{s_1 \doteq t_1, \dots, s_n \doteq t_n\}$);
9.  $\{t_1, \dots, t_m \,|\, k\} \doteq \{s_1, \dots, s_n \,|\, k'\}$:
    return SUA(unify_set($\{t_1, \dots, t_m\}, \{s_1, \dots, s_n\}$) $\cup \{k \overset{*}{=} k'\} \cup \mathcal{E}'$);
10. $\{s_1, \dots, s_n \,|\, k\} \doteq \{t_1, \dots, t_m \,|\, Z\}$:
    return SUA($\{\{t_1, \dots, t_m \,|\, Z\} \doteq \{s_1, \dots, s_n \,|\, k\}\} \cup \mathcal{E}'$);
11. $\{t_1, \dots, t_m \,|\, Z\} \doteq \{s_1, \dots, s_n \,|\, k\}$, choose n.d. $\emptyset \neq \{s_{l_1}, \dots, s_{l_k}\} \subseteq$
    $no\_dup(\{s_1, \dots, s_n\})$: return SUA($\{Z \overset{*}{=} no\_dup(\{s_1, \dots, s_n\} -$
    $\{s_{l_1}, \dots, s_{l_k}\} \cup Z' \cup k)\} \cup$
          limit_1($\{t_1, \dots, t_m\}, \{s_{l_1}, \dots, s_{l_k}\}, Z', \mathcal{E}')|_{FV([s_1, \dots, s_n, t_1, \dots, t_m, Z])}$);
12. $\{t_1, \dots, t_m \,|\, Z\} \doteq \{s_1, \dots, s_n \,|\, Z\}$: select n.d. between:
     i. choose n.d. $T \subseteq \{t_1, \dots, t_m\}$ and $S \subseteq \{s_1, \dots, s_n\}$ such that
        $T \cup S \neq \{t_1, \dots, t_m, s_1, \dots, s_n\}$: return SUA(unify_set($T, S$) $\cup \{Z \overset{*}{=}$
        $no\_dup((\{t_1, \dots, t_m\} - T) \cup (\{s_1, \dots, s_n\} - S) \cup N)\} \cup \mathcal{E}'$);
    ii. return SUA(unify_set($\{t_1, \dots, t_m\}, \{s_1, \dots, s_n\}$) $\cup \mathcal{E}'$);
13. $\{t_1, \dots, t_m \,|\, W\} \doteq \{s_1, \dots, s_n \,|\, Z\}$, $Z \not\equiv W$;
    choose n.d. $\{t_{p_1}, \dots, t_{p_j}\} \subseteq no\_dup(\{t_1, \dots, t_m\})$, $\{s_{l_1}, \dots, s_{l_k}\} \subseteq$
    $no\_dup(\{s_1, \dots, s_n\})$:
    return SUA($\{Z \overset{*}{=} no\_dup(\{s_1, \dots, s_n\} - \{s_{l_1}, \dots, s_{l_k}\} \cup Z')$,
    $W \overset{*}{=} no\_dup(\{t_1, \dots, t_m\} - \{t_{p_1}, \dots, t_{p_j}\} \cup W')\} \cup$limit_2($\{t_{p_1}, \dots, t_{p_j}\}$,
    $\{s_{l_1}, \dots, s_{l_k}\}$,$Z'$,$W'$,$\mathcal{E}')|_{FV([s_1, \dots, s_n, t_1, \dots, t_m, Z, W])}$).
```

Figure 4.12: A minimal set unification algorithm

```
        ii. return $\overset{2}{=} (t_i, \{t_1, \dots, t_m\}, \{s_1, \dots, s_n\})$;
        iii. return $\overset{3}{=} (t_i, \{t_1, \dots, t_m\}, \{s_1, \dots, s_n\})$;
    (b) if $Common\_part \neq \emptyset$
        then choose n.d. $S_0, S_1 \subseteq Common\_part$, $S_0 \cap S_1 = \emptyset$;
        choose n.d. $T_0 \subseteq Disagr_1$ and $T_1 \subseteq Disagr_2$ such that $|T_0| \geq |S_0|$
        and $|T_1| \geq |S_1|$: return unify_set($Disagr_1 - T_0, Disagr_2 - T_1$ ) $\cup$
        unify_set2($T_1, S_1$) $\cup$ unify_set2($T_0, S_0$).
```

Some comments on unify_set must be done in order to relate it to the unification problems (1), (2), (3) and (4). Action 4.(a) is for solving problems of type (1) and (2). The basic idea is that when an answer is computed by SUA without using the function $\overset{2}{=}$, then such answer may be considered as a surjective function from the leftmost set to the rightmost one. On the other hand, the use of $\overset{2}{=}$ is connected to the concept of *k-fork* presented in the analysis of problem (2). For this reason, action 4.(b) calls the new function unify_set2 which avoids the use of $\overset{2}{=}$, since (as commented in the analysis of problem (4)) to consider *k-forks* produces redundant solutions to problem (4) (the use of function $\overset{2}{=}$ for problem (3) always leads to failure).

```
   function unify_set2($\{t_1, \dots, t_m\}, \{s_1, \dots, s_n\}$);
1. if $m = 1$ and $n > 1$ then return $\{t_1 \doteq s_i : i = 1, \dots, n\}$;
2. if $m \geq 1$ and $n = 1$ then return $\{t_i \doteq s_1 : i = 1, \dots, m\}$;
3. if $m, n > 1$ then fix a value $i \in \{1, \dots, m\}$; select n.d. one of the following
   actions:
     i. return $\overset{1}{=} (t_i, no\_dup(\{t_1, \dots, t_m\}), no\_dup(\{s_1, \dots, s_n\}))$;
     ii. return $\overset{3}{=} (t_i, no\_dup(\{t_1, \dots, t_m\}), no\_dup(\{s_1, \dots, s_n\}))$.
```

$\overset{1}{=}$ matches one element t_i of the first set with one element s_j of the second one, and combines the two sets deprived of the selected elements. $\overset{1}{=}$ has the following structure:

```
function $\overset{1}{=}(t_i, \{t_1, \dots, t_m\}, \{s_1, \dots, s_n\})$;
choose n.d. one $j \in \{1, \dots, n\}$:
return $\{t_i \doteq s_j\} \cup$
    unify_set($\{t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_m\}, \{s_1, \dots, s_{j-1}, s_{j+1}, \dots, s_n\}$).
```

$\overset{2}{=}$ captures the concept of *k-fork* and is defined as follows:

```
function $\overset{2}{\doteq}(t_i, \{t_1, \ldots, t_m\}, \{s_1, \ldots, s_n\})$;
choose n.d. $S \subset \{s_1, \ldots, s_n\}$ such that $|S| \geq 2$:
return $\{t_i \doteq s$ : for all $s \in S\} \cup$
    unify_set($\{t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m\}, \{s_1, \ldots, s_n\} - S\}$).
```

$\overset{3}{\doteq}$ captures the concept of *k-cone* and its definition is:

```
function $\overset{3}{\doteq}(t_i, \{t_1, \ldots, t_m\}, \{s_1, \ldots, s_n\})$;
choose n.d. $\emptyset \neq T \subset \{t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m\}$:
return $\{t_i \doteq t$ : for all $t \in T\} \cup \overset{1}{\doteq}(t_i, \{t_1, \ldots, t_m\} - T, \{s_1, \ldots, s_n\})$.
```

The function limit_1 limits the possible values of Z, avoiding the introduction of variables $Y_j \in S$ when Y_j occurs in some *h-fork* for a solution to $\{X_1, \ldots, X_m\} \doteq S$ ($S \subseteq \{Y_1, \ldots, Y_n\}$—see analysis of problem (6)). The definitions of $\overset{1}{\doteq}$, $\overset{2}{\doteq}$, and $\overset{3}{\doteq}$ are embedded in the definition of limit_1; in this case extracting those subsets of S does not produce redundant solutions.

```
function limit_1($S_1$,$S_2$, $Z$,$\mathcal{E}$);
Let $\{t_1, \ldots, t_m\} = no\_dup(S_1)\,; \{s_1, \ldots, s_n\} = no\_dup(S_2)\,;$
1. If $\{t_1, \ldots, t_m\}$ and $\{s_1, \ldots, s_m\}$ are syntactically equal
   then return SUA($\{Z \doteq \emptyset\} \cup \mathcal{E}$);
2. elseif $m = 1$ and $n > 1$ then
   return SUA($\{s_i \doteq t_1 :\ 1 \leq i \leq n\} \cup \{Z \doteq \emptyset\} \cup \mathcal{E}$)
3. elseif $m = 1$ and $n = 1$ or $m > 1$ and $n = 1$ then choose n.d. $T \subseteq \{s_1\}$
   and return SUA($\{t_i \doteq s_1 :\ 1 \leq i \leq m\} \cup \{Z \doteq T\} \cup \mathcal{E}$)
4. else fix $i \in \{1, \ldots, m\}$ and choose n.d. one of the following actions:
   i. choose n.d. $j \in \{1, \ldots, n\}$ and $T \subseteq \{s_j\}$:
        return SUA($\{t_i \doteq s_j\} \cup \{Z \doteq T \cup Z'\} \cup$
       limit_1($\{t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m\}, \{s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_n\}, Z', \mathcal{E}$));
   ii. choose n.d. $S \subset \{s_1, \ldots, s_n\}$ such that $|S| \geq 2$:
       return SUA($\{t_i \doteq s :$ for all $s \in S\}$
        $\cup$ limit_1($\{t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m\}, \{s_1, \ldots, s_n\} - S, Z, \mathcal{E}$));
   iii. choose n.d. $T' \subset \{t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m\}$, $j \in \{1, \ldots, n\}$ and
       $T \subseteq \{s_j\}$:
       return SUA($\{t \doteq s_j :$ for all $t \in T'\} \cup \{t_i \doteq s_j\} \cup \{Z \doteq T \cup Z'\} \cup$
       limit_1($\{t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_m\} - T', \{s_1, \ldots, s_{j-1}, s_{j+1}, \ldots, s_n\}, Z', \mathcal{E}$)).
```

The definition of the function limit_2 is similar to the one of limit_1 but now the values of W and Z are constrained in order to avoid the introduction of variables occurring in some *h-fork* or *h-cone* respectively, of any solution θ to $S_0 \doteq S_1$, for some $S_0 \subseteq \{X_1,\dots,X_m\}$, $S_1 \subseteq \{Y_1,\dots,Y_n\}$—see analysis of problem (8). On the other hand, limit_2 must also control that those variables bounded in θ by a simple binding not be introduced in Z and W simultaneously. Like in limit_1, the definitions of $\overset{1}{=}$, $\overset{2}{=}$, and $\overset{3}{=}$ are embedded in the definition of limit_2.

```
function limit_2($S_1$,$S_2$, $Z$,$W$,$\mathcal{E}$);
Let $\{t_1,\dots,t_m\} = no\_dup(S_1)\,;\{s_1,\dots,s_n\} = no\_dup(S_2)\,;$
1. If $\{t_1,\dots,t_m\}$ and $\{s_1,\dots,s_n\}$ are syntactically equal then
   return SUA($\{Z \doteq \emptyset\} \cup \{W \doteq \emptyset\} \cup \mathcal{E}$);
2. elseif $m = 1$ and $n > 1$ then choose n.d. $T \subseteq \{t_1\}$:
   return SUA($\{s_i \doteq t_1 :\ 1 \le i \le n\} \cup \{Z \doteq T\} \cup \{W \doteq \emptyset\} \cup \mathcal{E}$);
3. elseif $n = 1$ and $m > 1$ then choose n.d. $S \subseteq \{s_1\}$:
   return SUA($\{t_i \doteq s_1 :\ 1 \le i \le m\} \cup \{Z \doteq \emptyset\} \cup \{W \doteq S\} \cup \mathcal{E}$);
4. elseif $m = 1$ and $n = 1$ then choose n.d. $T \subseteq \{t_1\}$, $S \subseteq \{s_1\}$ such that
   $T \cup S \neq \{t_1, s_1\}$: return SUA($\{t_i \doteq s_1 :\ 1 \le i \le m\} \cup \{Z \doteq T\} \cup \{W \doteq$
   $S\} \cup \mathcal{E}$);
5. else fix $i \in \{1,\dots,m\}$ and choose n.d. one of the following actions:
     i. choose n.d. $j \in \{1,\dots,n\}$ and $S \subseteq \{s_j\}$, $T \subseteq \{t_j\}$ such that
        $T \cup S \neq \{s_j, t_i\}$:
        return SUA($\{t_i \doteq s_j\} \cup \{W \doteq S \cup W'\} \cup \{Z \doteq T \cup Z'\} \cup$
        limit_2($\{t_1,\dots,t_{i-1},t_{i+1},\dots,t_m\}$,
              $\{s_1,\dots,s_{j-1},s_{j+1},\dots,s_n\}, Z', W'\mathcal{E}$));
    ii. choose n.d. $S \subset \{s_1,\dots,s_n\}$ such that $|S| \ge 2$, $T \subseteq \{t_i\}$:
        return SUA($\{t_i \doteq s :$ for all $s \in S\} \cup \{Z \doteq T \cup Z'\} \cup$
        limit_2($\{t_1,\dots,t_{i-1},t_{i+1},\dots,t_m\}, \{s_1,\dots,s_n\} - S, Z', W, \mathcal{E}$));
   iii. choose n.d. $T \subset \{t_1,\dots,t_{i-1},t_{i+1},\dots,t_m\}$, $j \in \{1,\dots,n\}$ and
        $S \subseteq \{s_j\}$:
        return SUA($\{t \doteq s_j :$ for all $t \in T\} \cup \{t_i \doteq s_j\} \cup \{W \doteq T \cup W'\} \cup$
        limit_2($\{t_1,\dots,t_{i-1},t_{i+1},\dots,t_m\} - T$,
              $\{s_1,\dots,s_{j-1},s_{j+1},\dots,s_n\}, Z, W', \mathcal{E}$)).
```

As already said, termination, correctness and completeness of SUA follow trivially by the results holding for the algorithm Unify_set presented in § 4.2.4. The very long and technical proof concerning its minimality for the presented sample problem is here omitted: it can be found in [10].

Part II

Logic Programming with Sets

Chapter 5

Constraints

The aim of this Chapter is to introduce techniques for simplifying constraints in the theories described in Chapter 3. Such simplification rewriting will return simple formulae whose satisfiability is easy to check. We are only interested here in simple constraints (conjunction of literals); it is in fact well-known (cf., i.e., Theorem 3.7, drawn from [16]) that the satisfiability problem for more complex 'set' constraints might be undecidable. In Chapter 6, dealing with negation, we will face the problem to extend the simplification of constraint techniques to more complex formulae.

The techniques to handle conjunctions of (positive) atoms based on the predicate symbol '$\doteq$' have been presented in Chapter 4. Here we will integrate such analysis with *negative* literals based on '$\doteq$' and with *positive* and *negative* literals based on the predicate symbol '$\in$'.

In particular, for the theory `WF_sets` of well-founded and hybrid sets, all the algorithms needed for a sound and complete integration with the CLP scheme are presented. The same results can be described for the other theories presented in a similar way.

Results of § 5.3 were presented, using slightly different notations, in [37, 38, 36], and, in a very similar form, in [33, 42].

5.1 Membership

The behavior of the membership predicate '$\in$' is regulated by axioms (K), (W), and by the well-foundedness axiom (F_3) in its various forms (cf. § 3.2).

(K) allows to state that a constraint of the form $t \in f(\cdots)$, where f is distinct from all the symbols introduced skolemizing axiom (W) (namely $[\cdot \,|\, \cdot]$, $\{\!\![\cdot \,|\, \cdot]\!\!\}$, $[\![\cdot \,|\, \cdot]\!]$, and $\{\cdot \,|\, \cdot\}$) is always unsatisfiable, hence equivalent to `false`. (Furthermore, for the same reason, the negative literal $t \notin f(\cdots)$ is always equivalent to `true`.)

(W) allows to saying that a constraint of the form $r \in [s \,|\, t]$ is equivalent to the disjunction $r \doteq s \vee r \in t$. (For the same reason $r \notin [s \,|\, t]$ is equivalent to the conjunction $r \not\doteq s \wedge r \notin t$.) The same holds also for $\{\!\![\cdot \,|\, \cdot]\!\!\}$, $[\![\cdot \,|\, \cdot]\!]$, and $\{\cdot \,|\, \cdot\}$.

In § 3.1.2 also the denumerable family of enumerated membership $\{\in^0, \in^1, \in^2, \ldots\}$, together with an axiomatization, consisting of the axioms $NW \in_* E^m$ aiming to give an alternative equality principle for bags is provided.

Under such axiomatization,

- the constraint $r \in^0 s$ is equivalent to $r \notin s$ (hence $r \notin^0 s$ is equivalent to $r \in s$);
- the constraint $r \in^{k+1} s$ is logically equivalent to

  $$\exists N \, (s \doteq \{\!\![\underbrace{r, \ldots, r}_{k+1} \,|\, N]\!\!\} \wedge r \notin N) \, .$$

  Therefore, it can be rewritten as $s \doteq \{\!\![\underbrace{r, \ldots, r}_{k+1} \,|\, N]\!\!\} \wedge r \notin N$, where N is a new variable.

  (In what follows any new variable is implicitly existentially quantified.)
- Its negative counterpart $r \notin^{k+1} s$ is logically equivalent to the infinite disjunction

  $$\bigvee_{i \neq k+1} r \in^i s \, ,$$

in its turn equivalent to the finite disjunction

$$\bigvee_{i \le k} r \in^i s \lor s \doteq \{\!\!\{ \underbrace{r, \dots, r}_{k+2} \mid N \}\!\!\} ,$$

where N is a new variable.

In the case of bags and sets, when we are interested in finite structures only or a removal principle is assumed, a constraint of the form $t \in X$, with X variable, can be explicitly re-written using equality as follows: $X \doteq \{\!\!\{ t \mid N \}\!\!\}$ ($X \doteq \{ t \mid N \}$), where N is a new generated variable.

In the case of well-founded theories we can also infer that the constraint $t[X] \in X$[1] is equivalent to `false` (hence $t[X] \notin X$ is equivalent to `true`).

5.2 Negated Equality

First we briefly observe that equality axiom ($\doteq_1$) ensures that for any variable X, $X \neq X$ is equivalent to `false`.

Freeness axiom (F_1) ensures that $f(s_1, \dots, s_n) \neq f(t_1, \dots, t_n)$—if f is different from $\{\!\!\{ \cdot \mid \}\!\!\}$, $[\![\cdot \mid \cdot]\!]$, and $\{ \cdot \mid \cdot \}$—is equivalent to the disjunction $\bigvee_{i=1}^n s_\imath \neq t_i$ (when $n = 0$, this reduces to the empty disjunction, namely, `false`).

Freeness axiom (F_2) allows to saying that the constraint

$$f(t_1, \dots, t_n) \neq g(s_1, \dots, s_m) ,$$

where f is different from g, is always `true`.

Freeness axioms (F_1) ensures that $f(s_1, \dots, s_n) \neq f(t_1, \dots, t_n)$, f different from $\{\!\!\{ \cdot \mid \}\!\!\}$, $[\![\cdot \mid \cdot]\!]$, and $\{ \cdot \mid \cdot \}$, is equivalent to the disjunction $\bigvee_{i=1}^n s_\imath \neq t_i$ (if $n = 0$ this is always `false`).

In the case of a constraint $s \neq t$, where s and t are based on the same interpreted functional symbol $\{\!\!\{ \cdot \mid \cdot \}\!\!\}$, $[\![\cdot \mid \cdot]\!]$, or $\{ \cdot \mid \cdot \}$, the *equality principle* must be call into play.

[1]As in § 3.2, $t[x]$ denotes a term having x as its proper subterm.

In the case of compact lists, axiom (E_k^c) (cf. § 3.1.3) states that

$$[\![\, t_1 \,|\, s_1 \,]\!] \not\doteq [\![\, t_2 \,|\, s_2 \,]\!]$$

is equivalent to the formula

$$\neg(t_1 \doteq t_2 \wedge (s_1 \doteq s_2 \vee s_1 \doteq [\![\, t_2 \,|\, s_2 \,]\!] \vee [\![\, t_1 \,|\, s_1 \,]\!] \doteq s_2)),$$

namely

$$t_1 \not\doteq t_2 \vee (s_1 \not\doteq s_2 \wedge s_1 \not\doteq [\![\, t_2 \,|\, s_2 \,]\!] \wedge [\![\, t_1 \,|\, s_1 \,]\!] \not\doteq s_2).$$

In the case of bags and sets, the introduction of a new (existentially quantified) entity in the equality principle causes a complexity problem in this kind of rewriting. For instance, in the bag case (cf. axiom (E_k^m)),

$$\{\!\{\, t_1 \,|\, s_1 \,\}\!\} \not\doteq \{\!\{\, t_2 \,|\, s_2 \,\}\!\}$$

is equivalent to

$$\neg((t_1 \doteq t_2 \wedge s_1 \doteq s_2) \vee \exists z\, (s_1 \doteq \{\!\{\, t_2 \,|\, z \,\}\!\} \wedge s_2 \doteq \{\!\{\, t_1 \,|\, z \,\}\!\})),$$

namely

$$(t_1 \not\doteq t_2 \vee s_1 \not\doteq s_2) \wedge \forall z\, (s_1 \not\doteq \{\!\{\, t_2 \,|\, z \,\}\!\} \vee s_2 \not\doteq \{\!\{\, t_1 \,|\, z \,\}\!\})).$$

The introduced universal quantification generates a constraint difficult to manipulate.

We need to use a different extensionality principle to handle both constraints of the form $\{\!\{\cdots\}\!\} \not\doteq \{\!\{\cdots\}\!\}$ and of the form $\{\cdots\} \not\doteq \{\cdots\}$.

We first solve the (finite) set case. Using classical extensionality axiom (E) extended with kernel entities (cf. § 3.1.4):

$$(E^s) \quad \forall v_1 v_2 \begin{pmatrix} \forall x\,(x \in v_1 \leftrightarrow x \in v_2) \wedge \\ ker(v_1) \doteq ker(v_2) \end{pmatrix} \rightarrow v_1 \doteq v_2\ ,$$

the constraint

$$\{s_1 \,|\, s_2\} \not\doteq \{t_1 \,|\, t_2\}$$

can be proved to be equivalent to the disjunction

$$\begin{array}{l} (\exists Z \in \{t_1 \,|\, s_1\})(Z \notin \{t_2 \,|\, s_2\}) \vee \\ (\exists Z \in \{t_1 \,|\, s_1\})(Z \notin \{t_2 \,|\, s_2\}) \vee \\ ker(s_1) \not\doteq ker(s_2) \end{array}$$

As shown in § 3.2, ker is a meta functional symbol that can be defined as follows:

$$\begin{array}{rcl} ker(f(\cdots)) & = & f(\cdots) \\ ker(\{s_1 \,|\, s_2\}) & = & ker(s_2)\ . \end{array}$$

Axiom (E^s) is implied by (E^s_k) in all models of finite sets (in particular in models whose domains are subsets of the Herbrand universe—cf. Theorem 3.33). We can hence use it without affecting the axiomatic theory. We only need to outline that in this way the constraint solver uses an 'internal' functor symbol that can help the expressiveness of the computed answer.

The *kernel* constraint makes sense only if Σ contains free functor symbols; if this is not the case, then the third disjunct is always `false`, and hence it can be removed. Notice that $ker(X)$ cannot be further simplified when X denotes a variable. In § 5.3 it will be shown in detail how to handle kernel constraints.

The rewriting of the bag case based on the same stream of ideas leads to the infinite (hence useless) disjunction

$$\begin{array}{l} \exists z \bigvee_{i>0}(z \in^i \{\!\!\{ t_1 \,|\, s_1 \}\!\!\} \wedge z \not\in^i \{\!\!\{ t_2 \,|\, s_2 \}\!\!\})\vee \\ \exists z \bigvee_{i>0}(z \in^i \{t_2 \,|\, s_2\} \wedge z \not\in^i \{t_1 \,|\, s_1\})\vee \\ ker(s_1) \neq ker(s_2) \end{array}$$

Luckily, there is the alternative equivalent rewriting:

$$\begin{array}{l} (t_1 \neq t_2 \wedge t_1 \notin s_2)\vee \\ (\{\!\!\{ t_2 \,|\, s_2 \}\!\!\} \doteq \{\!\!\{ t_1 \,|\, N \}\!\!\} \wedge s_1 \neq N) \end{array}$$

The correctness of this rewriting follows from the following

Lemma 5.1 *In any model of* `WF_bags` *or* `NWF_bags` *in which every element of the domain has finitely many occurrences of membership, then* $\{\!\!\{ t_1 \,|\, s_1 \}\!\!\} \neq \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}$ *if and only if* $(t_1 \neq t_2 \wedge t_1 \notin s_2) \vee \exists N\, (\{\!\!\{ t_2 \,|\, s_2 \}\!\!\} \doteq \{\!\!\{ t_1 \,|\, N \}\!\!\} \wedge s_1 \neq N)$

Proof. Let c be the constraint $\{\!\!\{ t_1 \,|\, s_1 \}\!\!\} \neq \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}$. Then c is equivalent to $(c \wedge t_1 \notin \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}) \vee (c \wedge t_1 \in \{\!\!\{ t_2 \,|\, s_2 \}\!\!\})$, namely $c \wedge$ `true`.

Since $t_1 \notin \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}$ implies c, then the first disjunct is equivalent to $t_1 \neq t_2 \wedge t_1 \notin s_2$.

If s_2 is a finite bag, then $t_1 \in \{\!\!\{ t_2 \,|\, s_2 \}\!\!\}$ is equivalent to

$$\exists N\, (\{\!\!\{ t_1 \,|\, N \}\!\!\} \doteq \{\!\!\{ t_2 \,|\, s_2 \}\!\!\})\ .$$

It remains to prove that the second disjunct is equivalent to $\exists N\,(\{\!\!\{\, t_2 \,|\, s_2 \,\}\!\!\} \doteq \{\!\!\{\, t_1 \,|\, N \,\}\!\!\} \wedge s_1 \not\doteq N)$. In one direction, assume that $s_1 \doteq N$. Then, applying substitution, we have that $\{\!\!\{\, t_1 \,|\, s_1 \,\}\!\!\} \doteq \{\!\!\{\, t_2 \,|\, s_2 \,\}\!\!\}$ and $\{\!\!\{\, t_1 \,|\, s_1 \,\}\!\!\} \not\doteq \{\!\!\{\, t_2 \,|\, s_2 \,\}\!\!\}$: a contradiction. The other direction follows immediately from the fact (`true` for bags) that $s_1 \not\doteq N$ implies $\{\!\!\{\, t_1 \,|\, s_1 \,\}\!\!\} \not\doteq \{\!\!\{\, t_1 \,|\, N \,\}\!\!\}$.

5.1 □

In the case of well-founded theories, there are other constraints whose behavior is regulated by axiom (F_3) in its various forms. $X \not\doteq t[X]$ is `true` unless

- $t[X]$ is $[\![\, t_0, \ldots, t_n \,|\, X \,]\!]$. In this case the above constraint is equivalent to $t_0 \not\doteq t_1 \vee \cdots \vee t_{n-1} \not\doteq t_n$,
- $t[X]$ is $\{t_0, \ldots, t_n \,|\, X\}$. In this case the above constraint is equivalent to the disjunction $\bigvee_{i=0}^{n} t_i \notin X$.

We end this section showing how negative equality and membership literals for sets can be interchanged:

$$\begin{array}{lcll} s \not\doteq t & \leftrightarrow & s \notin \{t\} & \text{This holds also for } [\cdot\,|\,\cdot], [\![\,\cdot\,|\,\cdot\,]\!], \{\!\!\{\,\cdot\,|\,\cdot\,\}\!\!\}. \\ t \notin s & \leftrightarrow & \{t \,|\, s\} \not\doteq s\,. & \end{array}$$

5.3 A full system for handling hybrid and well-founded set constraints

As sketched in the Introduction, the CLP scheme is a family of (Constraint) Logic Programming languages that become an effective language $CLP(\mathcal{X})$ when a particular domain of computation and constraints is assigned to the parameter $\mathcal{X}$. Effectiveness of the extended resolution procedure is ensured if the chosen domain satisfies a number of requirements. In this section we will recall such requirements and we will show that they are fulfilled by the domain of well-founded and hybrid sets. This allows to obtain a Constraint Logic Programming language with Sets.

Using a notation close to the one of [54], we will present the instance $CLP(\mathcal{S})$, where $\mathcal{S}$ is the 4-tuple $\langle\langle \Pi_C, \Sigma \cup \mathcal{V}\rangle, \mathcal{D}, \mathcal{L}, \mathcal{T}\rangle$, where

- $\Pi_C = \{\dot{=}, \in\}$ is the set of constraint predicate symbols,
- $\Sigma \supseteq \{\emptyset, \{\cdot \,|\, \cdot\}\}$ is the set of functional symbols;
- $\mathcal{V}$ is a denumerable set of variables ($\langle \Pi_C, \Sigma \cup \mathcal{V}\rangle$ together form the *signature*);
- $\mathcal{D} = \langle D, I\rangle$ is the $\langle \Pi_C, \Sigma\rangle$-structure defined in § 3.1, deeply analyzed in § 3.1.4 and denoted by $H_\Sigma/\equiv_s$ (the *initial model* of the equational theory $(E_1)(E_2)$);
- the class $\mathcal{L}$ of constraint formulae consists of all the conjunctions of $\langle \Pi_C, \Sigma \cup \mathcal{V}\rangle$-literals;
- $\mathcal{T}$ is the axiomatic theory of well-founded hybrid sets `WF_sets` presented in § 3.2.4, and consisting of the axioms $(\dot{=})$, (K), (W), (E_k^s), (F_1'), (F_2), and (F_3^s).

We briefly recall here the definition of the interpretation function I of the structure $H_\Sigma/\equiv_s$:

$$\begin{cases} a^I = a & \text{for any constant symbol } a \text{ in } \Sigma \\ (f(t_0,\dots,t_n))^I = f(t_0^I,\dots,t_n^I) & \text{for any } f \text{ in } \Sigma,\ ar(f) = n+1 \\ \{t_0,\dots,t_n \,|\, t\}^I = \{s_0,\dots,s_p \,|\, t^I\} & \text{where } t \text{ is a ur-element,} \\ & p \leq q, \text{ and} \\ & \text{every } s_i \text{ is } t_j^I \\ & \ \text{for some } j = 0,\dots,n, \text{ and} \\ & s_0 \prec s_1 \prec \cdots \prec s_p \end{cases}$$

where $\prec$ is an ordering relation over the Herbrand universe (for instance the one presented in § 3.1.4).

The first requirement that $\mathcal{S}$ must satisfy is the so-called *solution compactness* of the structure $\mathcal{D}$.

Definition 5.2 *A structure $\mathcal{D} = \langle D, I\rangle$ is said to be* SOLUTION COMPACT *if for all $a \in D$ the following hold:*

1. *'a' is uniquely definable by a finite or an infinite constraint (namely 'a' is the unique solution to such constraint);*

2. *if 'a' is a limit element (namely it is definable by an infinite constraint c_ω), then for any constraint c, $c_\omega \wedge c$ is not solvable if and only if there exists a finite constraint c_f such that $c \wedge c_f$ is not solvable.*

The interpretation of `WF_sets` in $H_\Sigma/\equiv_s$ is clearly *solution compact*, since each element $a \in H_\Sigma/\equiv_s$ is uniquely definable by the finite constraint $X \doteq a$. Moreover, since any element of $H_\Sigma/\equiv_s$ is finite, then no limit elements occur in $H_\Sigma/\equiv_s$.

Moreover, $\mathcal{T}$ and $\mathcal{D}$ must satisfy the further requirement:

Definition 5.3 *Given $\mathcal{S} = \langle\langle\Pi_C, \Sigma \cup \mathcal{V}\rangle, \mathcal{D}, \mathcal{L}, \mathcal{T}\rangle$, $\mathcal{D}$ and $\mathcal{T}$* CORRESPOND *if*

1. *$\mathcal{D} \models \mathcal{T}$, and*

2. *for any constraint c, $\mathcal{D} \models \exists c$ implies $\mathcal{T} \models \exists c$.*[2]

Lemma 5.4 `WF_sets` *and $H_\Sigma/\equiv_s$ correspond.*

Proof. First we show that $H_\Sigma/\equiv_s \models$ `WF_sets`, by proving that $H_\Sigma/\equiv_s \models \varphi$ for each φ axiom of `WF_sets`. The result is trivial for axioms $(\doteq)$, (F_1'), and (F_2).

(K) and (W): Immediate from the definition of $H_\Sigma/\equiv_s \models t^I \in^I s^I$ if and only if s^I is of the form $\{r_1, \ldots, r_n, t^I \mid r\}$.

(E_k^s): First we see that every situation of the following four (the rightmost formula of the axiom)

1. $y_1^I = y_2^I$ and $v_1^I = v_2^I$,
2. $y_1^I = y_2^I$ and $v_1^I = \{y_2 \mid v_2\}^I$,
3. $y_1^I = y_2^I$ and $\{y_1 \mid v_1\}^I = v_2^I$,
4. $v_1^I = \{y_2 \mid k\}^I$ and $v_2^I = \{y_1 \mid k\}^I$, for some k,

[2]With the notation $\exists c$ we denote the existential closure of c.

implies $\{y_1 \,|\, v_1\}^I = \{y_2 \,|\, v_2\}^I$. The first case is trivial. The second and the third follows from the fact that $\{t, t \,|\, s\}^I = \{t \,|\, s\}^I$, for any term t and s, by definition of I ($\mathcal{D}$ is a model of (E_2)). The fourth follows from the fact that $\{y_1, y_2, \,|\, k\}^I = \{y_2, y_1, \,|\, k\}^I$, always by definition of I ($\mathcal{D}$ is a model of (E_1)).

For the converse direction, assume that $\{y_1 \,|\, v_1\}^I = \{y_2 \,|\, v_2\}^I$; the result follows by the fact that a case analysis of the four cases generated by the combination of $(y_1^I = y_2^I \vee y_1^I \neq y_2^I) \wedge (v_1^I = v_2^I \vee v_1^I \neq v_2^I)$ always produces one of the disjuncts 1–4 above:

$y_1^I = y_2^I \wedge v_1^I \neq v_2^I$: Since $\{y_1 \,|\, v_1\}^I = \{y_2 \,|\, v_2\}^I$, this means exactly that $y_1^I = y_2^I \wedge ((v_1^I = \{y_2 \,|\, v_2\}^I \wedge (y_2^I \notin v_2^I)) \vee (\{y_1 \,|\, v_1\}^I = v_2^I \wedge (y_1^I \notin v_1^I))$;

$y_1^I \neq y_2^I$: This implies the fourth disjunct.

$y_1^I = y_2^I \wedge v_1^I = v_2^I$: This is the first disjunct. Moreover, if $y_1^I \in v_1^I$, it is also `true` that $v_1^I = \{y_2 \,|\, v_2\}^I$ and $\{y_1 \,|\, v_1\}^I = v_2^I$. Conversely, if $y_1^I \notin v_1^I$, also the fourth disjunct is implied.

The four results imply immediately the disjunct of the axiom we were looking for.

(F_3^s): If $H_\Sigma/\equiv_s \models t_1^I \in^I x^I \wedge \cdots t_n^I \in^I x^I$, then $H_\Sigma/\equiv_s \models \{t_1, \ldots, t_n \,|\, x\}^I = x^I$, by definition of I. It is easy to see that this is the only case of non monotonicity in the construction of ground terms.

It remains to show that $H_\Sigma/\equiv_s \models \exists c$ implies `WF_sets` $\vdash \exists\, c$, for each constraint c. It is sufficient to prove the claim for the atomical constraints:

(a) $H_\Sigma/\equiv_s \models \exists \bar{x}(s \doteq t)$,

(b) $H_\Sigma/\equiv_s \models \exists \bar{x}(s \not\doteq t)$,

(c) $H_\Sigma/\equiv_s \models \exists \bar{x}(s \in t)$,

(d) $H_\Sigma/\equiv_s \models \exists \bar{x}(s \notin t)$.

In order to show (a) we will prove the claim: *if $s^I = t^I$, then* `WF_sets` $\vdash s \doteq t$, by structural induction on the ground term s.

- s is a constant: $s^I = s$. This implies that $t^I = s (= t)$ and, by $(\doteq_1)$, we have that WF_sets $\vdash s \doteq t$.

- s is the ur-element $f(s_1, \ldots, s_n)$. This means that $f(s_1, \ldots, s_n)^I = f(s_1^I, \ldots, s_n^I) = t^I$, and that t is of the form $f(t_1, \ldots, t_n)$, where for each $i = 1, \ldots, n$, $t_i^I = s_i^I$. By induction hypothesis, for each $i \in \{1, \ldots, n\}$, it follows that WF_sets $\vdash s_i \doteq t_i$. Then, by $(\doteq_2)$, WF_sets $\vdash f(s_1, \ldots, s_n) \doteq f(t_1, \ldots, t_n)$.

- s is the set term $\{s_0, \ldots, s_m \,|\, h\}$, where h is a ur-element. This means that

  - $s^I = t^I = \{r_0, \ldots, r_p \,|\, r\}$, $p \leq m$, where the r_i's are pairwise distinct;
  - t has the form $\{t_0, \ldots, t_n \,|\, k\}$, $n \geq p$;
  - there exist two surjective functions

    $$\begin{array}{l} \pi_1 : \{0, \ldots, m\} \to \{0, \ldots, p\}\,, \text{ and} \\ \pi_2 : \{0, \ldots, n\} \to \{0, \ldots, p\} \end{array}$$

    such that: $s_i^I = r_{\pi_1(i)}$ and $t_j^I = r_{\pi_2(j)}$, for $i = 0, \ldots, m$ and for $j = 0, \ldots, n$.
  - Moreover, $h^I = k^I = r$.

  By induction hypothesis WF_sets $\vdash \bigwedge_{\pi_1(a)=\pi_2(b)} s_a \doteq t_b$ and WF_sets $\vdash h \doteq k$. The result follows from (E_k^s).

(a) follows immediately. (c) follows from (a) and the fact that $H_\Sigma / \equiv_s \models \exists \bar{x}\, (s \in t) \leftrightarrow \exists \bar{x} z\, (t \doteq \{s \,|\, z\})$ and WF_sets $\vdash \exists \bar{x}\, (s \in t) \leftrightarrow \exists \bar{x} z\, (t \doteq \{s \,|\, z\})$.

Likewise, to prove (b), we need the to prove the fact *if* $s^I \neq t^I$ *then* WF_sets $\vdash s \not\doteq t$ always by structural induction on the ground term s.

- The base case in which s is a constant, follows trivially by axiom (F_2').

- s is $f(s_1, \ldots, s_n)$, f different from $\{\cdot \,|\, \cdot\}$. This means that $s^I = f(s_1^I, \ldots, s_n^I)$. $s^I \neq t^I$ if and only if

- $t^I = g(r_1^I, \dots, r_m^I)$, f different from g, or
- $t^I = f(r_1^I, \dots, r_n^I)$, and there exists an index i such that $s_i^I \neq r_i^I$.

The result follows directly by axiom (F_2) in the first case and by axiom (F_1') and induction hypothesis in the second one.

- s is $\{s_0, \dots, s_m \,|\, h\}$, h ur-element. This means that: $s^I = \{r_0, \dots, r_p \,|\, r\}$, $p \leq m$, where the r_i's are pairwise distinct and there exists a surjective function $\pi : \{0, \dots, m\} \rightarrow \{0, \dots, p\}$ such that $s_i^I = r_{\pi_1(i)}$, for $i = 0, \dots, m$. $s^I \neq t^I$ if and only if

  1. $t^I = f(r_1^I, \dots, r_m^I)$, f different from $\{\cdot \,|\, \cdot\}$ (axiom (F_2), or
  2. $t^I = \{t_0^I, \dots, t_n^I \,|\, h^I\}$, h ur-element. Axiom (E_k^s) allows to infer that at least one of the following cases hold:
     - $r \neq h^I$, or
     - exists an element r_i distinct from all t_j^Is, or
     - exists an element t_j^I distinct from all r_is.

  In the former case the proof follows from axiom (F_2). In the latter, by induction hypothesis and (E_k^s).

This is sufficient for proving (b). To prove (d) it is sufficient to observe that

$$s \notin t \leftrightarrow t \neq \{s \,|\, t\}$$

holds both in the structure $H_\Sigma / \equiv_s$ and in the theory `WH_sets`.

5.4 □

The constraint solver

In order to produce a constraint solver for the class of formulae we are interested in, given a a conjunction of literals C, we split it into its components $C_{\doteq} \wedge C'_{\not\doteq} \wedge C''_{\not\doteq} \wedge C_{\in} \wedge C_{\notin} \wedge C^F$, where

- each C_π is a conjunction of literals of the form $s \pi t$;

- $C'_{\neq}, C''_{\neq}$ are respectively composed by atomic constraints not involving the symbol *ker* at all, and involving it at least once;

- C^F is empty (namely `true`) or `false`.

The constraint solver is an algorithm which verifies the solvability in the structure (which implies satisfiability in the theory because of Lemma 5.4) of a generic conjunction of $\langle \Pi_C, \Sigma \cup \mathcal{V} \rangle$-atoms. The initial constraint is successively transformed into an equi-satisfiable disjunctive normal form; each disjunct is in a simplified form for which the satisfiability is guaranteed.

Here below we describe the actions taken by the algorithm on the different components of the constraint C.

The equality constraints are handled by the unification algorithm presented in § 4.2.4.

We eliminate all membership atomic constraints by replacing them by equivalent equality atomic constraints, using the algorithm member described in Fig. 5.1.

A constraint $C_{\notin}$ is said to be in *canonical form* if any literal in it is of the form $t \notin X$, where X is a variable and X does not occur in t. Function notmember, described in Fig. 5.1, performs the canonical form reduction.

To handle $\neq$-constraints, we first deal with the constraints in $C'_{\neq}$, those in which *ker* does not appear. $C'_{\neq}$ is said to be in *canonical form* if any constraint in it is of the form $X \neq t$, X a variable not occurring in t. The algorithm notequal for such rewriting is described in Fig. 5.1.

A few words about constraints involving the functional symbol *ker* are in order. We require explicitly that they can not be introduced by the user but only by the action $(8.c)$ of notequal. Moreover,we fix their canonical form either as:

- $ker(X) \neq f(t_1, \ldots, t_n)$, where X is a variable, f is different from $\{\cdot \mid \cdot\}$ and from *ker* or

- $ker(X) \neq ker(Y)$, where X, Y are distinct variables.

```
member($C$)
while $C_{\in}$ is not empty do
(1)  $t \in f(\cdots) \wedge C \mapsto$ false
(2)  $t \in X \wedge C \mapsto X \doteq \{t \,|\, N\} \wedge C$
(3)  $t \in \{w \,|\, v\} \wedge C \mapsto$
         (3.a)  $t \in v \wedge C$
         (3.b)  $t \doteq w \wedge C$
return $C$.

notmember($C$)
while $C_{\notin}$ is in not in canonical form do
(1)  $t \notin \{s \,|\, r\} \wedge C \mapsto t \not\doteq s \wedge t \notin r \wedge C$
(2)  $t \notin f(\cdots) \wedge C \mapsto C$
(3)  $t[X] \notin X \wedge C \mapsto C$
return $C$.

notequal($C$)
while $C'_{\not\doteq}$ is not in canonical form do
(1)  $f(t_1, \ldots, t_n) \not\doteq g(s_1, \ldots, s_m) \wedge C$
     $f$ different from $g$                                   $\mapsto C$
(2)  $f(t_0, \ldots, t_n) \not\doteq f(s_0, \ldots, s_n) \wedge C \mapsto$
         (2.0)  $t_0 \not\doteq s_0 \wedge C$
          $\vdots$   $\vdots$
         (2.n)  $t_n \not\doteq s_n \wedge C$
(3)  $f \not\doteq f \wedge C \mapsto$ false
(4)  $X \not\doteq X \wedge C \mapsto$ false
(5)  $t \not\doteq X \wedge C \mapsto X \not\doteq t$
(6)  $X \not\doteq t \wedge C$
     $t$ is $f(\cdots)$ and
     $X$ occurs in it, or
     $t$ is $\{t_0, \ldots, t_n \,|\, t\}$ and
     $X$ occurs in $t_0, \ldots, t_n$                          $\mapsto C$
(7)  $X \not\doteq \{t_0, \ldots, t_n \,|\, X\} \wedge C \mapsto$
         (7.0)  $t_0 \notin X \wedge C$
          $\vdots$   $\vdots$
         (7.n)  $t_n \notin X \wedge C$
(8)  $\{s \,|\, r\} \not\doteq \{u \,|\, t\} \mapsto$
         (8.a)  $N \in \{s \,|\, r\} \wedge N \notin \{u \,|\, t\} \wedge C$
         (8.b)  $N \in \{u \,|\, t\} \wedge N \notin \{s \,|\, r\} \wedge C$
         (8.c)  if $\Sigma \supset \{\emptyset, \{\cdot \,|\, \cdot\}, ker\}$ then $ker(r) \not\doteq ker(t) \wedge C$
return $C$.
```

Figure 5.1: Handling of constraints in a hybrid set context

The constraint $C''_{\neq}$ is in *pre-normalized form* if all its atomic constraints are in canonical form; it is in *canonical form* if it is in pre-normalized form and $\mathsf{kernel\_sat}(C''_{\neq}) = \mathtt{true}$, where $\mathsf{kernel\_sat}$ is defined below:

```
kernel_analyzer($C$)
while $C''_{\neq}$ is not in pre-normalized form do
(1)  $ker(s) \neq \{\cdots\} \wedge C \quad \mapsto \quad C$
(2)  $\left.\begin{array}{r} ker(\{s\,|\,r\}) \neq t \wedge C \\ t \text{ is not of the form } \{\cdots\} \end{array}\right\} \mapsto \quad ker(r) \neq t \wedge C$
(3)  $\left.\begin{array}{r} \mathrm{ker}(f(t_1,\dots,t_n)) \neq t \\ f \text{ different from } \{\cdot\,|\,\cdot\}, \text{ and} \\ t \text{ is not of the form } \{\cdots\} \end{array}\right\} \mapsto \quad f(t_1,\dots,t_n) \neq t \wedge C$
(4)  $\left.\begin{array}{r} f(t_1,\dots,t_n) \neq ker(t) \\ f \text{ different from } ker \end{array}\right\} \mapsto \quad ker(t) \neq f(t_1,\dots,t_n) \wedge C$
(5)  $ker(X) \neq ker(X) \quad \mapsto \quad \mathtt{false}$
if kernel_sat($C''_{\neq}$)= true
then return $C$
else return false
```

The algorithm $\mathsf{kernel\_sat}$ tests the satisfiability of the constraint $C''_{\neq}$ in pre-normalized form. It is ensured whenever the signature contains an infinite number of constant symbols, or at least a functional symbol of arity greater than 0, distinct from $\{\cdot\,|\,\cdot\}$ and *ker* that $C''_{\neq}$ has solutions (if these are the cases, we are able to construct an infinity of different *kernels*). If the signature contains only a *finite* number of constant symbols, more than $\emptyset$, then satisfiability has to be checked. The algorithm $\mathsf{kernel\_sat}$, described below, perform such check suggesting also a possible solution.

```
kernel_sat($C$)
  if $\Sigma$ is infinite or $\exists g \in \Sigma$ s.t. $ar(g) > 0$ or $C$ is the empty constraint
  then return true
  else ($\Sigma = \{\{\cdot\,|\,\cdot\}, ker, \emptyset = a_0, a_1, a_2, \dots, a_n\}, n > 1$)
     Let $\{X_1,\dots,X_m\} = FV(C)$;
     Consider the non-oriented graph of nodes $V$ and edges $E$ such that
          $V = \{v_1,\dots,v_m,a_0,\dots,a_n\}$
          $\{v_i,v_j\} \in E$ if and only if $(ker(X_i) \neq ker(X_j)) \in C$
```

```
    $< v_i, a_j >\in E$ if and only if $(ker(X_i) \neq a_j) \notin C$
if $\exists$ an assignment $f$ from $\{v_1, \ldots, v_m\}$ to $\{a_1, \ldots, a_n\}$ s.t.
    1. $f(v_i) = a_j$ implies $\{v_i, a_j\} \in E$, and
    2. $\forall i, j \in \{1, \ldots, m\}, i \neq j$ no cycles of the form
        $\{v_i, f(v_i)\}, \{f(v_i), v_j\}, \{v_j, v_i\}$ occur
then return true
else return false.
```

In what follows we will prove that any conjunction of literals in the special form returned by the combination of the rewriting algorithms presented is satisfiable in the theory of hybrid well-founded sets. First some preliminary definition is needed.

A constraint C is in *canonical form* either if it is '`false`' or its components $C_{\doteq}, C_{\notin}, C'_{\neq}, C''_{\neq}$ are all in canonical form and $C_{\in}$ is empty. The function *rank*, defined as:

$$rank(x) = \begin{cases} 0 & \text{if } x \text{ is of the form } f(\cdots),\ f \not\equiv \{\cdot \mid \cdot\} \\ \max\{rank(s), 1 + rank(t)\} & \text{if } x \text{ has the form } \{t \mid, s\} \end{cases}$$

returns the 'depth' of a ground set. The function *find*, defined as:

$$find(x, t) = \begin{cases} \{0\} & \text{if } t \text{ coincides with } x \\ \emptyset & \text{if } t \text{ is } \emptyset \\ \{1 + n : n \in find(x, y)\} \cup find(x, s) & \text{if } t \equiv \{y \mid s\} \end{cases}$$

returns the set of 'depth' in which a given element x occurs in the set t.

These two functions will be used in the following proposition to find a suitable $H_\Sigma/\equiv_s$-solution to atomical constraints. For instance, consider the atomical constraint $x \neq \{y\}$. By definition, $find(y, \{y\}) = \{1\}$. Considering the integer equation $v_x \neq v_y + 1$, obtained by the inequation $x \neq \{y\}$, and picking up one solution (i.e. $v_y = 0, v_x = 2$) we define the substitution: $\sigma = [x/\{\{\emptyset\}\}, y/\emptyset]$ ($\{\{\emptyset\}\}$ has 'depth' 2, while $\emptyset$ has 'depth' 0). $(x \neq \{y\})\sigma$ is obviously `true` in the structure $H_\Sigma/\equiv_s$.

Lemma 5.5 *Let C be a constraint in canonical form different from* `false`*. Then C is $H_\Sigma/\equiv_s$-solvable and* `WF_sets`*-satisfiable.*

Proof. To start we assume that $C''_{\neq}$ is empty; successively we will show how to extend the proof to the general case. Let $C = C_{\doteq} \wedge C_{\notin} \wedge C_{\neq}$, and assume C_π is in canonical form for each π in $\{=, \notin, \neq\}$.

$C_{\dot{=}}$ has the form $X_1 \doteq t_1 \wedge \ldots \wedge X_m \doteq t_m$ and $\forall i = 1, \ldots, m$ X_i appears uniquely in $X_i = t_i$ and $X_i \notin FV(t_i)$. We define the substitution $\theta_1 = [X_1/t_1, \ldots, X_m/t_m]$. Clearly $H_\Sigma/\equiv_s \models \forall C_{\dot{=}}\theta_1$.

$C_{\notin}$ has the form $r_1 \notin Y_1 \wedge \ldots \wedge r_n \notin Y_n$ (Y_i does not occur in r_i) and $C_{\dot{\neq}}$ has the form $Z_1 \neq s_1 \wedge \ldots \wedge Z_o \neq s_o\}$ (Z_i does not appear in s_i). Let $W_1, \ldots, W_h$ be the variables occurring in $r_1, \ldots, r_n, s_1, \ldots, s_o$ other than $Y_1, \ldots, Y_n, Z_1, \ldots, Z_o$. Furthermore, let $\theta_2 = [W_1/\emptyset, \ldots, W_h/\emptyset]$, and let

$$C^I_{\notin} = \bigwedge_{\substack{\{\cdot\mid\cdot\} \text{ and } \emptyset \text{ are} \\ \text{the only functional symbols in } t}} (t \notin X) \; in \; C_{\notin}\theta_2$$

$$C^I_{\dot{\neq}} = \bigwedge_{\substack{\{\cdot\mid\cdot\} \text{ and } \emptyset \text{ are} \\ \text{the only functional symbols in } t, \\ \text{and } t \text{ is not a variable}}} (X \neq t) \; in \; C_{\dot{\neq}}\theta_2$$

Let $\bar{s} = max(\{size(t) : (t \notin X) \; in \; C_{\notin}\theta_2\} \cup \{size(t) : (X \neq t) \in C_{\dot{\neq}}\theta_2\})$ and let $V_1, \ldots, V_k$ be the variables occurring in $C_{\dot{\neq}}\theta_2 \wedge C_{\notin}\theta_2$ but not in $C^I_{\dot{\neq}} \wedge C^I_{\notin}$.

Let $\theta_3 = [V_1/\{\emptyset\}^{\bar{s}+1}, \ldots, V_k/\{\emptyset\}^{\bar{s}+k}]$ and let $\bar{r} = \bar{s} + k + 1$.

It is straightforward to prove that $H_\Sigma/\equiv_s \models \forall(C_{\notin} \setminus C^I_{\notin})\theta_2\theta_3$ and $\mathcal{A} \models \forall(C_{\dot{\neq}} \setminus C^I_{\dot{\neq}})\theta_2\theta_3$.[3]

Let $R_1, \ldots, R_j$ be the variables occurring in $C^I_{\notin} \wedge C^I_{\dot{\neq}}$. Let $n_1, \ldots, n_j$ be auxiliary variables. Build an integer disequation system E in the following way:

1. $E = \{n_i > \bar{r} : \forall i \in \{1, \ldots, j\}\} \cup$
   $\{n_{i_1} \neq n_{i_2} : \forall i_1, i_2 \in \{1, \ldots, j\}, i_1 \neq i_2\}$.

2. For each atomical constraint $(R_{i_1} \neq t) \; in \; C^I_{\dot{\neq}}$:
   $E = E \cup \{n_{i_1} \neq n_{i_2} + c : \forall i_2 \neq i_1, \forall c \in \mathit{find}(R_{i_2}, t)\}$

3. For each atomical constraint $(t \notin R_{i_1}) \; in \; C^I_{\notin}$:
   $E = E \cup \{n_{i_1} \neq n_{i_2} + c + 1 : \forall i_2 \neq i_1, \forall c \in \mathit{find}(R_{i_2}, t)\}$

[3]Here '\' operates on conjunctions of atoms, but considering them as sets.

To solve the problem of finding a solution to E is trivial (it is sufficient to choose arbitrarily big solutions satisfying the constraints). Let $\{n_1 = \bar{n}_1, \ldots, n_j = \bar{n}_j\}$ be a solution, define $\theta_4 = [R_i/\{\emptyset\}^{\bar{n}_i} : \forall i \in \{1, \ldots, k\}]$. Furthermore, let $\theta_5 = [X/\emptyset : X$ appears in $C\theta_1\theta_2\theta_3\theta_4]$, and $\theta = \theta_1\theta_2\theta_3\theta_4\theta_5$.

First observe that $C\theta$ is a ground constraint. Then we show that $H_\Sigma/\equiv_s \models C\theta$.

1. Pick $(X \doteq t) \in C$; since $X\theta_1$ coincides with $t\theta_1 = t$, $H_\Sigma/\equiv_s \models (X = t)\theta$;
2. Pick $(t \notin X) \in C$: three cases are possible:
   (a) if a symbol f different from $\{\cdot \mid \cdot\}$ occurs in t, then $H_\Sigma/\equiv_s$ cannot model the membership of $t\theta$ (in which f occurs) in $X\theta$ (term of the form $\{\emptyset\}^i$);
   (b) if t is one of the variables $W_1, \ldots, W_k$, then $t\theta = t\theta_2 = \emptyset$ cannot belong to $X\theta = \{\emptyset\}^i$ since $i > \bar{s} + k + 1 > 1$;
   (c) Otherwise, from the solution to the integer system E, we obtain
   $rank(t\theta) \neq rank(X\theta) - 1$.
3. Analogous considerations can be applied to the constraints of the form $X \neq t$.

By freeness axioms we get $rank(s) \neq rank(t) \rightarrow s \neq t$; if $C''_{\neq}$ is not empty, kernel_sat automatically supplies the elements to be used as kernels in the sets used to define the θ_i's substitutions. Then C is $H_\Sigma/\equiv_s$-solvable. By Lemma 5.4, C is `WF_sets`-satisfiable.

5.5 □

The canonical form is reached by executing the (non-deterministic) algorithm sat which computes the minimum fixpoint of the algorithm step, defined as follows:

$$\mathsf{step}(C) = \mathsf{kernel\_analyzer}(\mathsf{notequal}(\mathsf{notmember}(\mathsf{unify}(\mathsf{member}(C))))).$$

Hence, sat will have the form:

$$\mathsf{sat}(C) \quad = \quad \text{while } \mathsf{step}(C) \neq C \text{ do}$$

```
        $C :=$ step$(C)$;
    return $C$.
```

Lemma 5.6 *Whatever are the non-deterministic choices performed during the execution of* step *there exists* n *such that* $\mathsf{step}^{n+1}(C) = \mathsf{step}^n(C)$ *is a constraint in canonical form (by* $\mathsf{step}^n(C)$ *we mean the iteration* n *times of* step *on an input* C *- conjunction of* (Π_C, Σ)*-literalss).*

Proof. In the case of termination, the procedure returns a constraint in canonical form (otherwise one of the steps of the algorithm would be applicable). Termination of the algorithm is based on the termination of each single algorithm at any call. By introducing a measure K of structural complexity of the constraints relative to a specific predicate symbol, it is immediate that it decreases every time a new call of the algorithm is performed. These partial results are combined into a global termination proof.
(1) Each algorithm terminates at any call:

member. Assume $K = \sum_{(X \in t) \in C_\in} size(t)$; then, it decreases at every call;

unify. See Theorem 4.16;

notmember. Assume $K = \sum_{(X \notin t) \in C_\notin} size(t)$; then, it decreases at every call;

notequal. Let $K_1 = \sum_{(s \neq t) \in C'_{\neq}} size(s)$, $K_2 = \sum_{(s \neq t) \in C'_{\neq}} (size(s) + size(t))$; then the pair $\langle K_1, K_2 \rangle$, ordered by the lexicographic order, is the selected complexity, which decreases at each call.

kernel_analyzer. Let $K_1 = |\{(s \neq ker(t)) \in C''_{\neq}\}|$, $K_2 = \sum_{(s \neq t) \in C''_{\neq}} size(s)$. Due the peculiar form of the inequations containing ker, the selected complexity $< K_1, K_2 >$, with the lexicographic order decreases at each call.

(2) Suppose C_1 is returned by notmember (unify(member(C))).
Then, notequal(C_1) may introduce atomic constraints over predicates other than '$\neq$' only in the following cases:

(7) In the successive call, if X does not occur in t_i, the constraint is in canonical form and then no actions may be performed on it; otherwise the constraint is eliminated by step (3) of notmember;

(8) An atomic constraint of the form
$\{s_0, \ldots, s_m \mid h\} \neq \{t_0 \ldots, t_n \mid k\}$, where h, k are variables or ur-elements, is replaced, according to action (8.a), by the following atomic constraints:

$$Z \in \{s_0, \ldots, s_m \mid h\}, \qquad (i)$$
$$Z \notin \{t_0 \ldots, t_n \mid k\} \qquad (ii)$$

(case (8.b) is analogous to case (8.a); case (8.c) does not generate a possible increasing of the complexity. By applying member, the constraint *(i)* is replaced by one of the following:

- $Z = s_i$, for some $i \in \{0, \ldots, m\}$.
  Therefore, unify applies the substitution $[Z/s_i]$ (only in *(ii)*), and notmember deals with the atomical constraint $s_i \notin \{t_0 \ldots, t_n \mid k\}$. Then, notmember replaces the last constraint with: $s_i \neq t_0 \wedge \ldots \wedge s_i \neq t_n \wedge s_i \notin k$, where k is a variable and $s_i \notin k$ is in canonical form, or k is a ur-element (in this case such constraint will disappear). The new call of notmember will work on objects having a smaller *size* and this implies the full termination.
- $h = \{Z \mid N\}$, where h is a *variable.*
  Then, unify applies the substitution $[h/\{Z \mid N\}]$, making the constraint *size* bigger. This may be done a number of times less or equal to the number of occurrences of the variable h in $C'_{\dot{=}}$.

(3) The total termination follows from the termination of the algorithm obtained as follows:

1. $C_1 =$ notmember(unify(member(C)));
2. $C_2 = C_1\theta$, where $\theta = [X/\{Z_1^{(X)}, \ldots, Z_{|X|}^{(X)}\}]$, in which for every X occurring in $C_{1_{\dot{\neq}}}$, we denote by $|\,X\,|$ the number of occurrences of X in $C_{1_{\dot{\neq}}}$, $Z_j^{(v)}$ are new distinct variables;

3. $C_3 = \mathsf{notequal}(C_2)$.

The computation restarts from $\mathsf{step}(C_3)$; termination follows by the fact that the critical case is never generated in this way.

5.6 □

The termination of step, proved in Lemma 5.6, and the finiteness of the number of non deterministic choices generated by sat in correspondence of each call of step, ensure the finiteness of the number of constraints non-deterministically returned by sat. We may then state the following:

Lemma 5.7 *Let $C_1, \dots, C_k$ be the constraints non-deterministically returned by* $\mathsf{sat}(C)$*, and $N_1, \dots, N_m$ be the variables occurring in $C_1, \dots, C_k$ but not in C. Then*

$$\texttt{WF\_sets} \vdash C \leftrightarrow \exists N_1, \dots, N_m \textstyle\bigvee_{i=1}^{k} C_i.$$

Proof. It is sufficient to show that, for each atomical action of each algorithm for handling constraints, we have $\texttt{WF\_sets} \vdash C \leftrightarrow \exists(C_1 \vee \dots \vee C_n)$ where C is the analyzed atomical constraint and $C_1, \dots, C_n$ are the $n \geq 0$ constraints produced non-deterministically by the single action. This fact follows from the considerations of § 5.1 and § 5.2.

5.7 □

As corollary we get:

Corollary 5.8 *C is* WF_sets*-satisfiable if and only if there exists a non-deterministic choice such that* $\mathsf{sat}(C) \neq \texttt{false}$.

Corollary 5.9 WF_sets *is satisfaction complete, namely, for any constraint C (a conjunction of literals) it is possible to check whether its existential closure is derivable or not.*

Chapter 6

The CLP language {log} and the relation between intensional sets and negation

{log} is a language built over an instance of the general CLP scheme which supplies well-founded (extensional) set terms and a few basic set-theoretical operations. Using the notation introduced in § 5.3, that instance can be named as $CLP(\mathcal{S})$. The development of this kind of extension raises several interesting theoretical as well as practical problems (see for instance [42]). In the first part of this Chapter, first we will briefly analyze some of these problems, trying to motivate our solutions (§ 6.1), then we will introduce the basic syntactic entities of the language {log} (§ 6.2). In the second part of this Chapter (§ 6.3), we will describe how negation can be used to implement intensional set formers in the language {log}. We will show that negation as failure (cf. § 6.3.2) allows to deal with ground intensional sets only (it can be used, for instance, in Database applications); the set of programs that can be dealt with can be enlarged using constructive negation (cf. § 6.3.3, § 6.3.4, § 6.3.5, and § 6.3.6); nevertheless, undecidability results of set theory (§ 6.3.7) do not allow to freely use intensional set formers in {log} programs.

All the results of this Chapter can be found in [41]. More in detail, § 6.1 can be found also in [42]; the main result of § 6.3 can be found also in [37]; a preliminary study of § 6.3.3–6.3.6 can be found in [19].

6.1 Preamble

A considerable variety of proposals have been put forward in the area of the integration between logic-based programming paradigms and set theory. Most of these proposals, however, limit their expressivity to extensional sets.

For instance, various works have appeared recently dealing with the management of *set constraints* (see e.g. [6, 48, 62, 63]). In such works set constraints are inclusion relations between expressions denoting sets of *ground* terms over a signature. The aim of this stream of works is to provide effective tools for *set-based program analysis*.

Two examples of Constraint Logic Programming with finite and extensional sets are the languages Clps ([69]) and Conjunto ([46]). In particular, [69] uses a constraint satisfaction mechanism based on partial consistency techniques, whereas [46] extends the finite domain approach to allow variable domains to vary on the finite powerset of the Herbrand Universe. The aim of the authors of such papers is to provide languages with sets whose primary goal is efficiency. As a side effect, set terms allowed are very simple.

A step further towards a more 'intensional' approach to set definition can be found in [65, 66], where restricted universal quantification is introduced as extension of traditional SLD resolution and its semantics defined in such a way to allow its use as set-builder.

The only proposals have been presented in which the issue of dealing with intensional sets in a semantically clean way is directly tackled are $\mathcal{LDL}$ (cf., e.g., [15]), *Subset-assertion Programming* (cf., e.g., [55]) and its extension Sure ([56]).

In this Chapter we will show how intensional sets can be implemented using a logic language endowed with negation. As sketched in [15], also the inverse coding is feasible. Namely, if a set grouping capability is a 'built-in' of the language, negation can be removed from the program, as shown here below:

1. replace any occurrence of a nullary relational symbol p with $\tilde{p}(\emptyset)$, $\tilde{p}$ a new predicate symbol;

2. replace any negated occurrence of a n-ary relational symbol ($n > 0$) $\neg p(t_1, \ldots, t_n)$ in the body of a clause of the program with

$$[t_1, \ldots, t_n] \notin \{Y : Y \doteq [s_1, \ldots, s_n] \wedge p(s_1, \ldots, s_n)\}\,.$$

Although interesting, this translation requires an operational semantics for the set grouping construct, a problem less analyzed in literature with respect to the problem of handling negation. Moreover, as discussed in [18], to have a 'clean' semantics for the set grouping constructor is equivalent to introduce in the theory the full power of the separation axiom of set theory.

6.1.1 Which representation of sets?

In order to represent an *extensional* set $\{t_0, \ldots, t_n\}$, at least three alternatives are viable:

i. *union of singletons*, i.e. $\{t_0\} \cup \ldots \cup \{t_n\}$;

ii. *list-like*, i.e. $\{t_0 \,|\, \{t_1 \,|\, \ldots \{t_n \,|\, \emptyset\} \ldots\}\}$;

iii. $\{\}_n(t_0, \ldots, t_n)$.

Solution i requires three functional symbols to be introduced: $\emptyset$, of arity 0, $\{.\}$, of arity 1, and $\cup$, of arity 2. In a non trivial set theory (such as *ZF*), $\cup$ must be *Associative* (i.e., $A \cup (B \cup C) \doteq (A \cup B) \cup C$), *Commutative* (i.e., $A \cup B \doteq B \cup A$) and *Idempotent* (i.e., $A \cup A \doteq A$). Moreover $\emptyset$ is the *identity* element with respect to union (i.e., $A \cup \emptyset \doteq \emptyset \cup A \doteq A$).

Solution ii requires two functional symbols to be introduced: $\emptyset$, of arity 0, and $\{\cdot \,|\, \cdot\}$, of arity 2. Again in a significant set theory, $\{\cdot \,|\, \cdot\}$ must exhibit the *permutativity property*, and the *absorption property* presented in § 3.1.

Solution iii requires the introduction of an infinite signature, with a different set-constructor for each possible finite set cardinality. This approach has been adopted in [99]. In order to use this solution it is necessary to introduce a non trivial equational theory capable of specifying unifiability of set-terms with different main functors (like in $\{\}_3(X, Y, Z) \doteq \{\}_2(a, b)$).

Representation *ii* is quite common when dealing with sets in logic programming. It is used for instance in [66], in [15] (where $\{\cdot \mid \cdot\}$ is called *scons*) and also in the Gödel language [50, 51]. [57] uses the $\cup$ operator but actually its behavior is that of the $\{\cdot \mid \cdot\}$ operator of approach *ii*.

Representation *i*, on the contrary, is often used when dealing with the problem of set unification on its own (e.g. [22, 72]) where set unification is dealt with as a problem of *ACI-unification*. In [23], such approach is deeply analyzed, showing how, admitting the power of full boolean logic, boolean unification admits a unique (but complex to use) unifier.

A set term of approach *ii* can always be translated to a corresponding set term of approach *i*. The converse is not always true. For instance, the term $X \cup \{Y\} \cup Z$ has no correspondent representation in approach *ii*. Actually, by taking representation *ii* we are considering just a particular case of the general ACI with singleton unification problem. As Jayaraman and Plaisted note in [57]: *'Basically, this restriction permits iteration over the elements of a set, rather than iteration over the subsets of a sets. While some expressive convenience is sacrificed by this restriction, most practical cases are unaffected'.*[1]

Approach *iii* allows only to express set-terms with a known upper-bound to their cardinality. No partially specified sets can be described in this language.

For the language {log} ([37, 38, 42, 36]), and in this thesis, approach *ii* has been chosen. This was further motivated by the desire to provide a uniform (parametric) presentation of the axiomatic theories of lists, multi sets, compact lists, and sets.

However, note that a unification algorithm dealing with *set terms* will result NP-complete in all the approaches mentioned above (cf. § 4.1, or [38]).

[1] In [68] sets are represented as in approach *i*. However, since in this proposal set operations are evaluated only when applied to *ground sets*, problems arising when solving an equation such as $X \cup Y \cup Z \doteq \{a\} \cup \{b\} \cup \{c\}$—which would admit 343 independent solutions—are avoided 'a priori'. Instead, such an equation is considered as a constraint and possibly returned as part of the computed answer.

6.1.2 Which primitive operations/relations on sets?

We focus on the basic operations/relations on sets such as equality ($\doteq$), membership ($\in$), inclusion ($\subseteq$), strict inclusion ($\subset$), union ($\cup$), intersection ($\cap$) and difference ($\setminus$).

Let us assume that the language at hand is an Horn clause language augmented with set terms, represented as in approach *ii*. Unification needs to be extended in order to unify set terms respecting the properties of $\{\cdot \,|\, \cdot\}$ described above.

We are interested in establishing which of the operations on sets listed above need to be provided as primitive operations of this language and which, on the contrary, can be conveniently programmed in the language itself. The selection should be performed on the basis of a number of features of the resulting language, such as expressive power, effectiveness and efficiency.

The following basic predicates can be easily programmed in the considered language (set predicates are written in infix notation).

- Equality:

  $X \doteq X \leftarrow$

- Membership:

  $X \in \{X \,|\, Y\} \leftarrow$

- Inclusion:

  $\emptyset \subseteq Y \leftarrow$
  $\{X \,|\, V\} \subseteq Y \leftarrow$
  $\quad X \in Y \wedge V \subseteq Y.$

The considered language (i.e., HCL + extensional set terms + set unification) is powerful and simple. Nevertheless, (at least) the following two issues cannot be adequately tackled:

- *effectiveness*: if, for instance, the resolution algorithm is applied to the goal $\leftarrow A \subseteq \{a\}$ then an infinite SLD-tree is generated trying to compute the (sound) answers $[A/\emptyset]$, $[A/\{a\}]$, $[A/\{a, a\}]$, ... To solve the problem one would need to add the literal $X \notin V$ to the body of the second clause defining $\subseteq$;

- *expressive power*: other basic set-operations, such as $\notin$, $\neq$, $\cap$, $\subset$, $\setminus$ cannot be programmed in the present language unless some form of negation is introduced.

It turns out that adding either $\notin$ or $\neq$ as primitive operations to the considered language would suffice to solve all these problems without requiring full negation to be introduced in the language.

First, note that $\notin$ and $\neq$ can be easily defined one in term of the other:

$$A \neq B \leftarrow \quad A \notin \{B\} \qquad\qquad A \notin B \leftarrow \quad \{A \,|\, B\} \neq B.$$

Then, all the remaining basic operations on sets listed above can be easily and effectively programmed in the extended language, as shown in [42, 37].

While it is feasible in principle to have either $\neq$ or $\notin$ as the only primitive set operations of the language, efficiency considerations lead us to assume that our language provides both of them as primitive.

6.1.3 *CLP*($\mathcal{S}$)

Standard *CLP* notations and results ([53, 54] and § 5.3) are assumed hereafter. In particular, Σ and Π denote denumerable collections of function symbols and predicate symbols with their signatures, respectively. $\mathcal{V}$ denotes a denumerable set of variables. Moreover, $\Pi = \Pi_C \cup \Pi_B$ and $\Pi_C \cap \Pi_B = \emptyset$, where Π_C and Π_B are the sets of constraint predicate symbols and the set of user-defined symbols, respectively. $\tau(\Sigma \cup \mathcal{V})$ and $\tau(\Sigma)$ denote the set of terms and ground terms, respectively.

As noticed in previous sections, we would like to have a *CLP* language which is able to deal with extensional set terms as well as with standard Herbrand terms. Sets are represented using the signature $\{\emptyset, \{\cdot \,|\, \cdot\}, \ldots\}$ introduced in Chapter 3. Finally, we fix the set Π_C of constraint predicate symbols of the *CLP*-scheme to be $\{\in, \doteq\}$.

Here are two sample *CLP*($\mathcal{S}$) programs (the precise meaning of these programs will be clarified in the next section).

- Checking membership of an element to the set $\mathsf{Set1} \setminus \mathsf{Set2}$:

  $$\mathsf{in\_difference(X, Set1, Set2)} \leftarrow \mathsf{X} \in \mathsf{Set1} \wedge \mathsf{X} \notin \mathsf{Set2}.$$

- Selecting subsets of S composed of even numerals only (standard definition of even is assumed):[2]

```
even_in(S, ∅) ←
even_in(S, {A | R}) ←
    A ∈ S ∧ A ∉ R □
    even(A), even_in(S, R).
```

6.2 The Language {log}

As mentioned at the beginning of this Chapter, our purpose is to define on top of $CLP(\mathcal{S})$ a more sophisticated language, named {log}, capable of a higher level of abstraction. {log} is basically $CLP(\mathcal{S})$ enriched with *intensional set formers* and *Restricted Universal Quantifiers* (*RUQ*s). The set of constraint predicates Π_C is still fixed to $\{\doteq, \in\}$. As shown in § 6.1.2, this set of primitive set-theoretic operations is sufficient to define most of the common set operations (such as union, intersection, ...).

Let us call $\tau(\Sigma \cup \mathcal{V})$-terms, $(\Pi_C, \Sigma \cup \mathcal{V})$-literals, and $(\Pi_B, \Sigma \cup \mathcal{V})$-literals, *simple terms*, *simple literal constraints*, and *simple literals*, respectively.

Definition 6.1 *We define an* Intensional Term *to be either*

- $\{X : c \,\square\, B_1, \ldots, B_n\}$ *such that* $X \in FV(c \,\square\, B_1, \ldots, B_n)$*, where* c *is a conjunction of simple literal constraints and* $B_1, \ldots, B_n$ *are simple literals;*

- $f(t_1, \ldots, t_n)$ *such that* $\exists i, 1 \leq i \leq n,\ t_i$ *is an intensional term,* $f \in \Sigma$.[3]

[2] Following standard *CLP* notation, the symbol '□', denoting conjunction, is used to separate the *constraint* part from the rest of the goal.

[3] We have already pointed out our desire to restrict attention to *finite sets*. Guaranteeing finiteness in the case of intensional sets requires that predicates in inten-

*A {log}-*TERM *is either a* simple *or an* intensional *term.*
*A {log}-*ATOM *is an atom $p(t_1, \dots, t_n)$, where $t_1, \dots, t_n$ are {log}-terms and $p \in \Pi_B$.*

For example:

- $\mathsf{f}(\mathsf{a}, \{\mathsf{5}\})$ is a simple term (not a set term);
- $\{\mathsf{X} : \mathsf{X} \neq 1 \,\Box\, \mathsf{p}(\mathsf{X})\}, \mathsf{f}(\mathsf{Y}, \{\mathsf{Z} : \mathsf{Z} \in \mathsf{Y} \,\Box\, \mathsf{p}(\mathsf{Z}, \mathsf{W})\})$ are intensional terms.

Definition 6.2 *A {log}-*EXTENDED LITERAL *is either*

- *a {log}-*LITERAL, *i.e. a positive or negative {log}-atom;*
- *a* RUQ-LITERAL $(\forall X_1 \in t_1) \dots (\forall X_n \in t_n)(c \,\Box\, B_1, \dots, B_m)$, *$n \geq 1$, $m \geq 0$, where c is a conjunction of simple literal constraints, $B_1, \dots, B_m$ is a finite sequence of simple literals, X_j's are pairwise distinct variables, and t_i's are* extensional terms *such that $X_j \cap FV(t_i) = \emptyset$ for $i \leq j$.*

In order to maintain the translation of *RUQ*s and intensional sets as simple as possible (see § A.4 and § 6.3, respectively), we do not allow nesting of *RUQ*s and intensional definitions, i.e. no *RUQ*s and intensional terms may appear inside other *RUQ*s and intensional terms. These restrictions could be easily relaxed through a suitable enhancement of the translation procedures.

A few words about *RUQ*-literals are in order. First, recall from § 2.1 that

$(\forall x \in t)\varphi,$

φ any first order formula, is a shorthand for

$\forall x\,(x \in t \rightarrow \varphi).$

In [38] the equivalence between {log} programs containing Restricted Universal Quantifications and {log} programs which are *RUQ*-free is

sional set formers define finite relations. However, the problem of ascertaining the finiteness of a predicate is known to be undecidable in the general case. In this context we rely on the programmer's attention to avoid generation of infinite sets (as done also in [66], for instance).

proved. Such proof can be also found in § A.4. Each occurrence of a *RUQ* may be removed by performing a simple syntactic translation. Thanks to this, we can assume from now on that programs we are working on do not contain any *RUQ*.

Definition 6.3 *A {log}-*CLAUSE *is a normal clause* $A{:}{-}\,c\Box B_1,\ldots,B_n$*, where A is a positive {log}-literal, c is a conjunction of simple atomic constraints, and* $B_1,\ldots,B_n$ *are {log}-extended literals. A {log}-*GOAL *is a {log}-clause with empty head. A {log}-*PROGRAM *is a finite set of {log}-clauses.*[4]

Notice that, following PROLOG syntax, {log} clauses use the ' :– ' symbol to distinguish the head from the body of the clauses. $CLP(\mathcal{S})$, following CLP syntax, uses the '←' symbol.

Here are two sample {log} programs involving intensional set terms (the precise meaning of the programs will be clarified in next sections).

- Intersection of two sets S1 and S2:

  $\mathsf{intersection(S1,S2,S3)} :- \mathsf{S3} \doteq \{\mathsf{X} : \mathsf{X} \in \mathsf{S1},\ \mathsf{X} \in \mathsf{S2}\}.$

- The set of prime numbers less than a given limit N:

  $\mathsf{primes(N,S)} :- \mathsf{S} \doteq \{\mathsf{X} : \mathsf{between(1,N,X)},\ \mathsf{is\_prime(X)}\}$
  $\mathsf{is\_prime(X)} :- \mathsf{S} \doteq \{\mathsf{Y} : \mathsf{between(1,X,Y)}\} \Box (\forall \mathsf{Z} \in \mathsf{S})(\mathsf{non\_div(Z,X)})$
  $\mathsf{between(A,B,C)} :- (\mathsf{A} < \mathsf{C}), (\mathsf{C} < \mathsf{B}).$
  $\mathsf{non\_div(A,B)} :- 0\ =\backslash=\ (\mathsf{B\ mod\ A}).$

Given a {log}-goal $:-\,c\Box\bar{B}$ and a {log}-program P, they are rewritten (cf. § 6.3) into a $CLP(\mathcal{S})$ goal $\leftarrow\, c' \Box \bar{B}'$ and a $CLP(\mathcal{S})$ with negation program P'. The operational semantics of the language {log} is therefore based on the standard CLP (plus negation) operational semantics (cf., e.g., [53]).

More in detail, given a $CLP(\mathcal{S})$ program (without negation) P and a $CLP(\mathcal{S})$ goal $G \equiv\leftarrow c \Box A_1,\ldots,A_n$, there are two main forms of derivations:

[4]As a notational convenience we will write $\{X : B_1,\ldots,B_n\}$ and $A{:}{-}\,B_1,\ldots,B_n$ instead of $\{X : c\Box B_1,\ldots,B_n\}$ and $A :-\, c\Box B_1,\ldots,B_n$, respectively, whenever c is `true`.

Algebraic derivation: a goal of the form $\tilde{G} \equiv\leftarrow \tilde{c} \,\Box\, \tilde{B}_1, \dots, \tilde{B}_n$ is a $H_\Sigma/\equiv_s$-resolvent of G in P if exist n variants of clauses of P

$$\tilde{A}_1 \leftarrow c_1 \,\Box\, \tilde{B}_1, \dots, \tilde{A}_n \leftarrow c_n \,\Box\, \tilde{B}_n,$$

such that

$$\tilde{c} \equiv c \wedge \textstyle\bigwedge_{i=1}^{n} \tilde{c}_i \wedge \bigwedge_{i=1}^{n} args(A_i \doteq \tilde{A}_i),$$

where $args(p(t_1, \dots, t_n) \doteq p(s_1, \dots, s_n))$ is defined to be the conjunction of equalities $t_1 \doteq s_1 \wedge \dots \wedge t_n \doteq s_n$, and $c \cup \tilde{c}_1 \cup \dots \cup \tilde{c}_n$ is satisfiable in $H_\Sigma/\equiv_s$.

Logic derivation: a goal of the form $\tilde{G} \equiv\leftarrow (\tilde{c} \,\Box\, \tilde{B}_1, \dots, \tilde{B}_n)\theta$ is a `WF_sets`-resolvent of G in P if exist n variants of clauses of P

$$\tilde{A}_1 \leftarrow c_1 \,\Box\, \tilde{B}_1, \dots, \tilde{A}_n \leftarrow c_n \,\Box\, \tilde{B}_n$$

such that

$$\tilde{c} \equiv c \wedge \textstyle\bigwedge_{i=1}^{n} \tilde{c}_i \wedge \bigwedge_{i=1}^{n} args(A_i \doteq \tilde{A}_i)$$

can be split in two parts $\tilde{c}_1 \wedge \tilde{c}_2$ fulfilling the following requirements:

- $\tilde{c}_2$ is a conjunction of positive literals (equations and membership literals; θ is a $(E_1)(E_2)$-unifier of $t \in s$ if and only if θ is a $(E_1)(E_2)$-unifier of $s \doteq \{t \mid s\}$) and it admits a complete set S of $(E_1)(E_2)$-unifiers (namely if γ is a $(E_1)(E_2)$-unifier of $\tilde{c}_2$, then exists $\delta \in S$ such that $\gamma \preceq_{\mathcal{T}} \delta$);
- $\theta \in S$;
- $\tilde{c}_1\theta$ is `WF_sets`-satisfiable.

In [53] it is proved that, provided the axiomatic theory (in this case `WF_sets`) and the structure (in this case $H_\Sigma/\equiv_s$) are related (as, in this case, they are), the two forms of derivations return the same success set. The conditions needed are the following:

- the solution compactness of the domain of the structure (cf. Def. 5.2);

- the theory and the structure must correspond (cf. Def. 5.3).

Solution compactness of $H_\Sigma / \equiv_s$ has been proved in the considerations following Def. 5.2; correspondence between `WF_sets` and $H_\Sigma / \equiv_s$ is proved in Lemma 5.4.

Concerning with the implementation of the two forms of derivations (using constraint solving and backtracking),

- the *algebraic derivation* is therefore algorithmically implementable using the function `sat`, described in § 5.3, to test the satisfiability of the constraint.

- To implement the *logic derivation* is sufficient to choose one of the $\tilde{C}$ constraints different from `false` returned by `sat`(C) and as θ the substitution induced by $\tilde{C}_{\doteq}$.

6.3 Compiling intensional sets

In [37, 19] it is argued that intensional sets can be programmed in a logic language with sets like $CLP(\mathcal{S})$, provided the language supplies either a *set grouping* mechanism or some form of *negation* in goals and clause bodies. This allows us, on the one hand, to consider intensional sets as a syntactic extension to be dealt with a simple preprocessing phase, and, on the other hand, not to be concerned with intensional sets when defining the semantics of the language. In this section we assume that all intensional sets refer to pure sets (namely sets built adding elements to the empty set $\emptyset$). It is easy (but, perhaps, not so useful) to extend the translation to colored sets; this makes all the translation steps syntactically heavier.

Let us try, first of all, to understand why $\neq$ and $\notin$ are not sufficient for a satisfactory definition of a set grouping mechanism, and full negation is required instead. By exploiting the extensionality axiom (E) (equivalent to (E^s_k) for our purposes—cf. § 3.1.4), one can prove that:

$$\begin{array}{rcl} \{x : p(x)\} \doteq S & \leftrightarrow & \forall x\,(x \in S \leftrightarrow p(x)) \\ & \leftrightarrow & \forall x\,(x \in S \to p(x)) \,\wedge\, \forall x (p(x) \to x \in S). \end{array}$$

As we can see, a set grouping feature requires the ability to perform *restricted universal quantification* as well as universal quantification over the solutions to an arbitrary predicate. Even though $CLP(\mathcal{S})$ can express restricted universal quantification, it is unable to express the other form of quantification.

$$\begin{array}{ll} \forall X(p(X) \rightarrow X \in S) & \leftrightarrow \\ \forall X(\neg p(X) \vee X \in S) & \leftrightarrow \\ \neg\exists X(X \notin S \wedge p(X)). & \end{array}$$

where the existentially quantified formula can be rendered in $CLP(\mathcal{S})$ through a new clause involving a local variable X (cf. the definition of $\mathsf{partial}_{\mathsf{pdiv}}$ shown below). Thus, what we really need is just some form of negation in clause bodies.

The correlation between set grouping and negation is furtherly illustrated by the following example. Suppose that, given a natural number N, we want to define a predicate returning the greatest prime number P in its decomposition in prime factors. We use an intensional set former to collect the prime divisors of N as follows:

$$\begin{array}{lll} \mathsf{max}_{\mathsf{pdiv}}(\mathsf{N},\mathsf{P}) & :- & \mathsf{max}(\{\mathsf{X} : \mathsf{pdiv}(\mathsf{N},\mathsf{X})\},\mathsf{P}) \\ \mathsf{max}(\mathsf{S},\mathsf{Max}) & :- & \mathsf{Max} \in \mathsf{S} \,\Box\, (\forall \mathsf{X} \in \mathsf{S})(\mathsf{Max} \geq \mathsf{X}) \\ \mathsf{pdiv}(\mathsf{A},\mathsf{B}) & :- & \mathsf{div}(\mathsf{A},\mathsf{B}),\ \mathsf{is\_prime}(\mathsf{B}) \end{array}$$

where $\mathsf{is\_prime}(\mathsf{B})$ is `true` if B is a prime number and div represents the divisibility relation.

In order to compute $\mathsf{max}_{\mathsf{pdiv}}$ we should be able to collect the set of all prime divisors computed by the predicate pdiv and, at the same time, to reject any partial solution, namely any proper subset of the set of all possible solutions to pdiv. This could be obtained by defining a new predicate, $\mathsf{setof}_{\mathsf{pdiv}}$, which explicitly performs the set-collection operation:

$$\begin{array}{lll} \mathsf{max}_{\mathsf{pdiv}}(\mathsf{N},\mathsf{P}) & :- & \mathsf{setof}_{\mathsf{pdiv}}(\mathsf{S},\mathsf{N}),\ \mathsf{max}(\mathsf{S},\mathsf{P}) \\ \mathsf{setof}_{\mathsf{pdiv}}(\mathsf{S},\mathsf{N}) & :- & (\forall \mathsf{X} \in \mathsf{S})(\mathsf{pdiv}(\mathsf{N},\mathsf{X})), \neg\mathsf{partial}_{\mathsf{pdiv}}(\mathsf{S},\mathsf{N}) \\ \mathsf{partial}_{\mathsf{pdiv}}(\mathsf{S},\mathsf{N}) & :- & \mathsf{X} \notin \mathsf{S} \,\Box\, \mathsf{pdiv}(\mathsf{N},\mathsf{X}). \end{array}$$

The replacement of intensional set terms by the setof predicates, which allow the corresponding extensional sets to be constructed, is performed by a two steps program transformation process. This process will transform a given {log} program into the equivalent $CLP(\mathcal{S})$ program with negation, where the set Π_B of predicate symbols of the given program is replaced by the set Π'_B containing Π_B and all the new predicate symbols required to implement intensional sets, e.g., the $\mathsf{setof}_{\mathsf{pdiv}}$ mentioned in the example (along with the new predicate symbols generated by translation of *RUQ*s).

The first step of this program-to-program transformation leads the {log} source code to a *normal form* where all variable instantiations in clauses and goals are expressed as constraints and each *discriminant* $(c \,\Box\, \bar{B})$ of intensional terms is expressed by a unique predicate symbol. Such a predicate symbol has arity equal to $|\, FV(c \,\Box\, \bar{B})\,|$ and is defined by a unique clause having the corresponding discriminant as its body.

Step 1 - Program normalization

Let C be the {log} clause

$$p(s_1, \ldots, s_m) \mathrel{:-} c \,\Box\, A_1(t^1_1, \ldots, t^1_{n_1}), \ldots, A_r(t^r_1, \ldots, t^r_{n_r})$$

where s_i's and t^i_j's are {log} terms, and $A_i(t^i_1, \ldots, t^i_{n_i})$ are {log} literals (as it ensues from the discussion in § 6.2, there is no need here to consider *RUQ*-literals).
Repeatedly perform the following actions until none applies.

- Replace C by the equivalent clause

$$\begin{array}{lcl} p(X_1, \ldots, X_m) & \leftarrow & c \wedge \bigwedge_{i=1}^m (X_i \doteq s_i) \wedge \bigwedge_{i=1}^r \bigwedge_{j=1}^{n_i} (X^i_j \doteq t^i_j) \,\Box \\ & & A_1(X^1_1, \ldots, X^1_{n_1}), \ldots, A_r(X^r_1, \ldots, X^r_{n_r}), \end{array}$$

  where X_i's and X^i_j's are new distinct variables.[5]

- Replace each atomic constraint $s\,\pi\,t$, where $\pi \in \Pi_C$ and s and/or t are intensional terms, with the constraint

$$s'\,\pi\,t' \wedge \bigwedge_{i=1}^m (S_i \doteq s_i) \wedge \bigwedge_{j=1}^n (T_j \doteq t_j),$$

[5]Observe that the above clause might be not a $CLP(\mathcal{S})$ clause, due to the presence of intensional sets and/or restricted universal quantifiers. However, due to the preprocessing computation, if this is the case, it will be removed soon from the program. For this reason we accept, in this context, the improper use of the symbol $\leftarrow$.

where s_i's and t_j's are all the intensional set terms occurring in s and t respectively, s' and t' are the extensional terms obtained by replacing the intensional set terms s_i's and t_j's in s and t with the new variables S_i's and T_j's respectively.

- Replace each atomic constraint of the form $Set \doteq \{X : c \,\Box\, \bar{B}\}$ by the constraint

  $$Set \doteq \{X : \delta(X, Z_1, \dots, Z_m)\},$$

  where $\{X, Z_1, \dots, Z_m\} = FV(c \,\Box\, \bar{B})$ and δ is a newly generated predicate symbol, and add to the program the new clause

  $$\delta(X, Z_1, \dots, Z_m) \leftarrow c \,\Box\, \bar{B}.$$

Step 2 - Elimination of intensional set terms

The second step is intended to remove intensional set terms from a normalized program according to the general idea for implementing set grouping sketched at the beginning of this section. For each predicate symbol δ generated by the normalization step above, two new predicate symbols, $setof_\delta$ and $partial_\delta$, are introduced, and their corresponding $CLP(\mathcal{S})$ definitions added to the generated program. The whole process is organized as follows:

- Replace each normalized clause of the form

  $$h(\bar{Y}) \leftarrow c \wedge Set_1 \doteq \{X : \delta(X, \bar{Z})\} \,\Box\, \bar{B}$$

  by the set of clauses

  $$\begin{array}{lcl} h(\bar{Y}) & \leftarrow & c \,\Box\, setof_\delta(Set_1, \bar{Z}), \bar{B} \\ setof_\delta(Set_1, \bar{Z}) & \leftarrow & (\forall X \in Set_1)\delta(X, \bar{Z}), \neg partial_\delta(Set_1, \bar{Z}) \\ partial_\delta(Set_1, \bar{Z}) & \leftarrow & X \notin Set_1 \,\Box\, \delta(X, \bar{Z}). \end{array}$$

For example, the definition of the predicate $\mathsf{max_{pdiv}}$ shown above is first replaced by the following clauses:

$$\begin{array}{lcl} \mathsf{max_{pdiv}}(\mathsf{N}, \mathsf{P}) & \leftarrow & \mathsf{Set_1} \doteq \{\mathsf{X} : \delta(\mathsf{X}, \mathsf{N})\} \,\Box\, ; \mathsf{max}(\mathsf{Set_1}, \mathsf{P}) \\ \delta(\mathsf{X}, \mathsf{N}) & \leftarrow & \mathsf{pdiv}(\mathsf{N}, \mathsf{X}). \end{array}$$

Then (second step), the normalized definition of the predicate $\mathsf{max}_{\mathsf{pdiv}}$ is replaced by:

$$\mathsf{max}_{\mathsf{pdiv}}(\mathsf{N},\mathsf{P}) \quad\leftarrow\quad \mathsf{setof}_{\delta}(\mathsf{Set}_1,\mathsf{N}),\mathsf{max}(\mathsf{Set}_1,\mathsf{N})$$

adding the clauses defining $\mathsf{setof}_{\delta}(\mathsf{Z},\mathsf{N})$ to the transformed program.

6.3.1 Intensional sets and Negation

In the previous section we have pointed out the close correlation between the set grouping mechanism, needed to support intensional sets, and the availability of negation in clause bodies, needed to define the set grouping mechanism in the language itself. It is well-known that different forms of negation can be introduced in logic programming [9]. Here we focus our attention on two of the most commonly considered forms of negation: *negation as failure* [97] and *constructive negation* [25, 26]. First we discuss the problems arising when negation as failure is used to implement intensional sets. Then we illustrate the problems and results obtained by integrating (an extended version of) constructive negation in the $CLP(\mathcal{S})$ framework.

We would like to point out that negation in $CLP(\mathcal{S})$ will be considered exclusively as a tool to implement intensionally defined sets—we will not deal with general and unrestricted uses of negation.

6.3.2 Negation as Failure

Negation as failure (NAF) [28] is the best known form of negation adopted in logic programming. It is based on the simple rule

$$P \not\vdash A \rightarrow P \vdash \neg A$$

(although it is practically instantiated as negation as *finite* failure—i.e. $P \vdash \neg A$ if A belongs to the finite failure set of P [97]). Negation as failure has the valuable advantage of simplicity, which easily leads to devise efficient implementations for it. Conversely, soundness and completeness of the resolution procedure extended with negation as failure (i.e., SLDNF-resolution) can be obtained only by applying suitable syntactical restrictions to programs and goals (cf., for instance, [9]). In particular, the *allowedness* property—that is the property that every variable in a clause occurs in at least one positive literal—can be used as a sufficient condition for guaranteeing soundness of the SLDNF-procedure with respect to the completed version of a program. In fact soundness of SLDNF requires the selection rule to process negative literals only if they are *ground*. Allowedness guarantees that the computation does not *flounder*, i.e. it does not reach a situation in which all literals are negative and none of them is ground.

In order to have a sound use of negation as failure in {log}, a viable approach is to adapt the notion of *allowedness* to {log} programs [39]. Using this modified notion of allowedness, it is possible to show that all the results related to soundness and completeness of SLDNF-resolution still hold in the {log} framework.

In doing this, one must properly take into account the presence of set constraints as well as of *RUQ*-literals and intensional sets, and the way they are dealt with via translation to $CLP(\mathcal{S})$ programs. Unfortunately, the translation of *RUQ*s generates non-allowed clauses. For example, given the following definition of the subset relation

$\mathsf{subset}(\mathsf{S1}, \mathsf{S2}) :-$
    $(\forall \mathsf{X} \in \mathsf{S1})(\mathsf{X} \in \mathsf{S2})$

the generated $CLP(\mathcal{S})$ code will look like (see § A.4)

$\mathsf{subset}(\emptyset, \mathsf{S2}) \leftarrow$
$\mathsf{subset}(\{\mathsf{A} \mid \mathsf{R}\}, \mathsf{S2}) \leftarrow$
    $\mathsf{A} \notin \mathsf{R} \wedge \mathsf{A} \in \mathsf{S2} \,\square\, \mathsf{subset}(\mathsf{R}, \mathsf{S2})$

where the first clause is obviously non-allowed. Similar problems come out with intensional sets, which are implemented using *RUQ*s.

The development of suitable syntactic restrictions for {log} programs and goals, by adapting those usually applied to conventional logic programs, though feasible, is far from being satisfactory. Indeed it seems to overly restrict the class of programs that can be practically used.

An alternative approach can be devised by taking advantage of the fact that, as mentioned above, negation is used exclusively to translate intensional set definitions. Thus, in a {log} program, once intensional sets have been removed—using the procedure described in § 6.3—negative literals appear only in clauses of the form:

$$setof_{\delta}(Set_1, \bar{Z}) \leftarrow (\forall X \in Set_1)\delta(X, \bar{Z}), \neg partial_{\delta}(Set_1, \bar{Z}).$$

Since we are adopting a LDNF-based resolution, the literal

$$\neg partial_{\delta}(Set_1, \bar{Z})$$

will not be selected as long as the computation of the immediately preceding RUQ has been completed. In this context, soundness of the resolution procedure is guaranteed whenever all the collected sets related to intensional definitions are *finite* and *ground*, and all free variables possibly occurring in such definitions (i.e., the set of variables represented as $\bar{Z}$) are grounded by the set grouping operations. For example, the definition of intersection given in § 6.2, even if non-allowed, can be safely executed whenever sets $S1$ and $S2$ are ground terms. Thus it is possible to extend the class of admissible programs by replacing syntactic restrictions on programs by simpler dynamic tests on sets constructed from intensional definitions.

The ability to collect finite and ground sets only, though limiting, is nevertheless sufficient for a large class of applications (e.g., deductive databases). In fact, most of the logic languages dealing with sets described in the literature present such a restriction.

In [41] it is shown that the intensional set formers of the language $\mathcal{LDL}$ ([15, 83]), as well as the set-collection capability of the *Subset-Assertion Programming* language ([55, 57]), the two most interesting approaches of this area, can be implemented in {log} with negation as failure, since the groundness restriction are practically similar.

6.3.3 From Negation as Failure to Constructive Negation

Aiming at the development of a general-purpose language with sets (as opposed to a language for some specific application area, e.g. deductive databases), we should try to relax as much as possible the restriction for collected sets to be ground. For instance, the following is a {log} program requiring the ability to collect a non-ground set:

```
r(Y, S) :-
    S ≐ {X : X ≐ f(V, Y) □ q(V)}
q(a).
q(b).
```

The goal

```
:- r(Y, S)
```

should return the constraint

$$\mathsf{S} \doteq \{\mathsf{f(a, Y)}, \mathsf{f(b, Y)}\}\,.$$

associating to S a set which is *non-ground*, although finite. Actually, the set-collection operation implied by the intensional set definition occurring in the given program is performed by executing a subgoal of the form

$$\leftarrow (\forall X \in S)(X \doteq f(V, Y) \,\Box\, q(V)), \neg partial(S, Y).$$

Because of the non-groundness of the sets returned by the restricted universal quantifier (e.g. $\{f(a, Y), f(b, Y)\}$), it is not possible to obtain the desired solution using SLDNF-resolution (some larger and more meaningful examples requiring collections of non-ground sets are presented at the end of § 6.3.6).

The approach we take to generalize the framework at hand is based on the use of *constructive negation* [12, 25, 43] as an alternative to negation as failure. In constructive negation the negative literal is not required to be ground and instantiations of the variables appearing in it are computed to allow the negative literal to be satisfied (from here comes the *constructive* nature of this approach).

The basic idea behind constructive negation goes as follows. Given a program P, the set $\sigma_1, \ldots, \sigma_n$ of all the solutions to a goal $\leftarrow G$ (assuming there are finitely many) is such that $Comp(P) \models G \leftrightarrow \sigma_1 \vee \cdots \vee \sigma_n$ where $Comp(P)$ is the completed version of the program P. Given a program P, the completion of P, $Comp(P)$ is the theory consisting of the 'closure' of the clauses in P (denoted *iff*(P) in [8]) plus a first order theory regarding the signature (equality plus freeness axioms in 'standard' logic programming, `WF_sets` in our approach). Basically, if a predicate p is defined by the k clauses

$$p(X_1, \cdots, X_n) \leftarrow B_i \quad 1 \leq i \leq k,$$

where each B_i contains an explicit existential quantification of the local variables, then the corresponding clause in the completion of P is

$$\forall X_1 \cdots X_n (p(X_1, \cdots, X_n) \leftrightarrow B_1 \vee \cdots \vee B_k).^6$$

Taking the negation of the formula, $Comp(P) \models \neg G \leftrightarrow \neg(\sigma_1 \vee \cdots \vee \sigma_n)$, gives an idea of how to obtain a solution to a negative literal in the constructive negation approach. The key point is the development of an effective procedure to extract actual solutions from the negation $\neg(\sigma_1 \vee \cdots \vee \sigma_n)$. The description given by Chan [25, 26] is an instantiation of this framework to the case of pure logic programming (each σ_i is a simple substitution). The relations between constructive negation and CLP have been studied in [104].

As we will see later on in this chapter, the integration of a form of constructive negation in the {log} framework is feasible, and allows considerable generalizations of the set-collection capabilities of the language (in particular, it will enable us to give a satisfactory solution to the sample program and goal discussed at the beginning of this section). Nevertheless, a full integration between sets and constructive negation does not appear to be possible, leading to some situations which are argued to be undecidable in § 6.3.7.

6.3.4 Constructive Negation: an overview

Let us start considering an overview of constructive negation [43, 93], adapted to our language and notation. Given a program P and a goal

$$\leftarrow C \,\Box\, \neg p(x_1, \ldots, x_n),$$

assume that $\sigma_1, \ldots, \sigma_m$ are all (assumed to be finite) the answer constraints. Each σ_i can be assumed to have the form (see [19] for further details)

$$\exists v_1^{(i)} \ldots v_{k_i}^{(i)} (c_1^{(i)} \wedge \ldots \wedge c_{h_i}^{(i)} \wedge d_1^{(i)} \wedge \ldots \wedge d_{l_i}^{(i)}), \qquad k_i, h_i, l_i \geq 0,$$

where $v_1^{(i)}, \ldots, v_{k_i}^{(i)}$ are those variables in σ_i other than $x_1, \ldots, x_n$, $c_j^{(i)}$'s are *positive* atomic constraints, and $d_j^{(i)}$'s are simple formulae of the

6 For an overview of the general properties of *iff*(P) and (standard) $Comp(P)$ see [8, 97].

form $\forall Y_1 \dots Y_m(c)$, c a *negative* atomic constraint. Moreover the $c_j^{(i)}$'s and the $d_j^{(i)}$'s are in a simplified canonical form,[7] which can be obtained by removing all the *redundant variables* and *equalities* and all the *irrelevant negative constraints*, according to the following rewriting rules:

$$\left.\begin{array}{r} (x \doteq z) \wedge C \\ x \in \{x_1,\dots,x_n\} \text{ and } z \notin \{x_1,\dots,x_n\} \end{array}\right\} \mapsto C^{\{z \mapsto x\}}$$

$$\left.\begin{array}{r} (z \doteq t) \wedge C \\ z \notin \{x_1,\dots,x_n\} \end{array}\right\} \mapsto C$$

$$\left.\begin{array}{r} (\forall Y_1 \dots Y_m(s \not\doteq t)) \wedge C \\ \Big(\big(FV(\forall Y_1 \dots Y_m(s \not\doteq t) \setminus \{x_1,\dots,x_n\}\big)\cap \\ FV(c_1,\dots,c_{h_i})\Big) \neq \emptyset \end{array}\right\} \mapsto C.$$

Soundness and completeness results can be used to show that

$$(1) \qquad Comp(P) \models \forall x_1 \dots x_n(p(x_1,\dots,x_n) \leftrightarrow \sigma_1 \vee \dots \vee \sigma_m).$$

Taking the negation of this formula,

$$Comp(P) \models \forall x_1 \dots x_n(\neg p(x_1,\dots,x_n) \leftrightarrow \neg\sigma_1 \wedge \dots \wedge \neg\sigma_m),$$

one can obtain $\neg p(x_1,\dots,x_n)$ by considering $\neg\sigma_1 \wedge \dots \wedge \neg\sigma_m$ where each $\neg\sigma_i$ can be rewritten as

$$(2) \qquad \forall v_1^{(i)} \dots v_{k_i}^{(i)}(\neg c_1^{(i)} \vee \dots \vee \neg c_{h_i}^{(i)} \vee \neg d_1^{(i)} \vee \dots \vee \neg d_{l_i}^{(i)}).$$

Chan's solution (cf. [25]; for a deeper analysis of the equality problem in Logic Programming see [98])—which is intended to apply to standard logic programming, where equality is interpreted as the usual syntactical equality over Herbrand terms—makes use of the following result:

Lemma 6.4 [Chan] *Given a universally quantified disjunction of the form*

[7]We are assuming here that the definition of constraint in canonical form is extended in order to encompass for simple formulae of the form $\forall Y_1 \dots Y_m(c)$, c a negative atomic constraint (see [19]).

$$\forall \bar{X}\bar{Y}(y \neq t \vee D)$$

where t is a term, D is a formula, $\bar{X}$ are variables in t, and $\bar{Y}$ are the variables in D not occurring in t, then

$$\models \forall y(\forall \bar{X}\bar{Y}(y \neq t \vee D) \leftrightarrow (\forall \bar{X}(y \neq t) \vee \exists \bar{X}(y \doteq t \wedge \forall \bar{Y} D))).$$

This Lemma would allow us to split the disjunction (2) in two cases, one in which the first disjunct ($\neg c_1^{(i)}$) is actually true for every instantiation of the universally quantified variables, and another one in which it is possible to detect an instantiation in which $\neg c_1^{(i)}$ fails (and as consequence $c_1^{(i)}$ succeeds) and where the universal quantification of the remainder of (2) must succeed.

By repeatedly applying this Lemma (along with distributivity of $\wedge$ over $\vee$) each $\neg\sigma_i$ can be transformed into the equivalent formula

$$(3) \qquad \begin{array}{rcl} \gamma_i & \equiv & (\forall \bar{Y}_1 \neg c_1^{(i)}) \vee \\ & & \exists \bar{Y}_1\, (c_1^{(i)} \wedge \forall \bar{Y}_2 \neg c_2^{(i)}) \vee \\ & & \quad \vdots \qquad\quad \vdots \\ & & \exists \bar{Y}_1 \cdots \bar{Y}_{h_i-1}\, (c_1^{(i)} \wedge \ldots \wedge c_{h_i-1}^{(i)} \wedge \forall \bar{Y}_{h_i} \neg c_{h_i}^{(i)}) \vee \\ & & \exists \bar{Y}_1 \cdots \bar{Y}_{h_i} (c_1^{(i)} \wedge \ldots \wedge c_{h_i}^{(i)} \wedge d_1^{(i)}) \vee \\ & & \quad \vdots \qquad\quad \vdots \\ & & \exists \bar{Y}_1 \cdots \bar{Y}_{h_i} (c_1^{(i)} \wedge \ldots \wedge c_{h_i}^{(i)} \wedge d_{l_i}^{(i)}), \end{array}$$

where $\bar{Y}_1, \ldots, \bar{Y}_{h_i}$ is a partition of the variables $v_1^{(i)}, \ldots, v_{k_i}^{(i)}$ accordingly to Lemma 6.4. Therefore,

$$\neg\sigma_1 \wedge \ldots \wedge \neg\sigma_m \leftrightarrow \gamma_1 \wedge \ldots \wedge \gamma_m.$$

One can further transform the formula $\gamma_1 \wedge \ldots \wedge \gamma_m$ by applying distributivity, and by removing existential quantifiers by suitably renaming variables and moving the quantifiers in front of each disjunct. By doing this, we obtain the equivalent formula in disjunctive normal form

$$\theta_1 \vee \theta_2 \vee \ldots \vee \theta_s$$

where $s \leq \Pi_{k=1}^{i}(h_k + l_k)$, and each θ_j is a conjunction of possibly universally quantified literals coming from the γ_i. To sum up,

$$Comp(P) \models \forall x_1 \ldots x_n(\neg p(x_1, \ldots, x_n) \leftrightarrow \theta_1 \vee \ldots \vee \theta_s).$$

Thus, each θ_j represents one of the s possible solutions to the given goal $\neg p(x_1, \ldots, x_n)$.

Observe that the above rewriting is based on two fundamental properties:

- the first, stated in Lemma 6.4, that ensures the distributivity of $\vee$ with respect to $\forall$;
- the second, guaranteed by the elimination of irrelevant negative constraints, which ensures that, after h_i applications of distributivity, universally quantified variables disappear.

As we will show later, the first property does not always hold in `WF_sets`. However, whenever it holds, the second property is a direct consequence of it.

From an operational point of view the transformation described so far amounts, for each given goal of the form $\leftarrow C \,\Box\, \neg p(x_1, \ldots, x_n)$, to explore non-deterministically s different alternatives

$$\leftarrow C \wedge \theta_1, \cdots, \leftarrow C \wedge \theta_s$$

applying to each of them the constraint satisfiability test with respect to the selected constraint theory. This requires the ability to deal with simple atomic constraints (e.g. $x \neq t$) as well as with universally quantified literals, i.e. simple formulae of the form $\forall \bar{X}(X \neq t)$ (inequality formulae can also be re-written in this form, using the fact $t \notin s$ if and only if $s \neq \{t \,|\, s\}$).

6.3.5 Constructive Negation in {log}

In the context of {log}, the use of constructive negation amounts, first of all, to extend the various constraint satisfiability procedures used by `sat` (cf. § 5.3) in order to accommodate for the new kind of simple universally quantified formulae resulting from the transformations described in the previous section (see [19] for a description of this extension).

Unfortunately, as we will discuss in details hereafter, Chan's Lemma does not always hold in the framework of the theory `WF_sets`.

To precisely realize this, we exhibit a case in which the $\leftarrow$ part of the implication of Lemma 6.4 does not hold.[8]

Let us consider the following {log} clause

$$\begin{array}{l} p(U) \text{ :-} \\ \quad U \doteq \{X_1, X_2\} \wedge X_1 \not\doteq \emptyset \end{array}$$

and the {log} goal

$$\text{:-}\ R \doteq \{X : X \in Y \,\Box\, p(Y)\}$$

which translates to

$$\leftarrow R \doteq S \,\Box\, \mathit{setof}_p(S, Y)$$

where

$$\begin{array}{l} \mathit{setof}_p(S, Y) \leftarrow \\ \quad (\forall X \in S)(X \in Y \,\Box\, p(Y)), \\ \quad \neg \mathit{partial}_p(S, Y). \end{array}$$

The subsequent goal is therefore

$$\leftarrow R \doteq S \,\Box\, (\forall X \in S)(X \in Y \,\Box\, p(Y)), \neg \mathit{partial}_p(S, Y).$$

The first solution to the left-hand part of this goal is $R \doteq \emptyset$. Therefore the new goal becomes

$$\leftarrow R \doteq \emptyset \,\Box\, \neg \mathit{partial}_p(\emptyset, Y).$$

Solving the goal $\leftarrow \mathit{partial}_p(\emptyset, Y)$ will require to solve the goal

$$\leftarrow X \not\in \emptyset \wedge X \in Y \,\Box\, p(Y)$$

which returns the two solutions

$$\begin{array}{l} \exists X X_1 X_2\, (X \doteq X_1 \wedge Y \doteq \{X_1, X_2\} \wedge X_1 \not\doteq \emptyset) \\ \exists X X_1 X_2\, (X \doteq X_2 \wedge Y \doteq \{X_1, X_2\} \wedge X_1 \not\doteq \emptyset) \end{array}$$

After removing redundant variables, they become the same answer

$$\exists X_1 X_2\, (Y \doteq \{X_1, X_2\} \wedge X_1 \not\doteq \emptyset).$$

Therefore, the answer to $\leftarrow \neg \mathit{partial}_p(\emptyset, Y)$ will be

$$(*) \quad \forall X_1 X_2\, (Y \not\doteq \{X_1, X_2\} \vee X_1 \doteq \emptyset).$$

[8]Conversely, the $\rightarrow$ part of this implication is a simple theorem of first order predicate calculus.

If Chan's Lemma would hold, then $(*)$ could be transformed to the equivalent formula

$$\begin{array}{ll}(**) & \forall X_1 X_2\,(Y \neq \{X_1, X_2\}) \vee \\ & \exists X_1 X_2\,(Y \doteq \{X_1, X_2\} \wedge X_1 \doteq \emptyset).\end{array}$$

Taking $Y = \{\emptyset, \{\emptyset\}\}$ then $(*)$ is `false`, whereas $(**)$ is `true`.

The reason why Chan's Lemma is failing in our framework is that this Lemma is implicitly founded on the assumption of the existence of a unique most general unifier for any unification problem. Unfortunately this situation may occur in our framework (as shown by the last example above), since in general a {log} unification problem may admit more than one distinct solution. This is a reason why Chan's approach (and other similar approaches) to constructive negation cannot be directly applied to the {log} framework.

6.3.6 Generalizing Chan's Result

In this section we introduce a new rewriting technique which will allows us to generalize Chan's approach to the {log} framework and, more in general, to any framework in which the unification problem admits finitely many solutions. This is obtained by the following lemma which generalizes Chan's Lemma.

Lemma 6.5 *Given a term t, and a formula D, let $\bar{X}$ be variables in t and $\bar{Y}$ variables in D not occurring in t. Let $\bar{X}'$ be $\mid \{\bar{X}\} \mid$ new variables. Let ρ be the substitution $[\bar{X}/\bar{X}']$ and assume the unification problem $t \doteq t^\rho$ admits in $(E_1)(E_2)$ exactly n most general unifiers $\theta_1, \ldots, \theta_n$. Assume moreover that, for $i = 1, \ldots, n$, $dom(\theta_i) = \{\bar{X}'\}$ and $ran(\theta_i) \subseteq \{\bar{X}\}$ (i.e. no new variable is introduced by unification), and that the variables of t not in $\bar{X}$ nor in $\bar{Y}$—if any—do not occur in the θ_i's. Then*

$$\texttt{WF\_sets} \models \quad \forall y \left(\begin{array}{l} \forall \bar{X}\bar{Y}(y \neq t \vee D) \leftrightarrow \\ \forall \bar{X}(y \neq t) \vee \exists \bar{X} \left(y \doteq t \wedge \bigwedge_{i=1}^{n} \forall \bar{Y} D^{\rho\theta_i} \right) \end{array} \right).$$

Proof. It is easy to see that $\forall \bar{X}\bar{Y}(y \neq t \vee D)$ is equivalent to

$$\forall \bar{X}(y \neq t) \quad \vee \quad \exists \bar{X}(y \doteq t \wedge \underbrace{\neg\exists \bar{X}'\bar{Y}(t \doteq t^\rho \wedge \neg D^\rho)}_{(*)}).$$

Thanks to the assumptions of the lemma. the formula $(*)$ is equivalent to

$$\neg\exists \bar{X}'\bar{Y}\left(\left(\bigvee_{i=1}^n \hat{\theta}_i\right) \wedge \neg D^\rho\right)$$

where $\hat{\theta}_i$ is the equational representation of the substitution θ_i. Applying distributivity the last formula becomes

$$\neg\bigvee_{i=1}^{n}(\exists \bar{X}'\bar{Y}\,\hat{\theta}_i \wedge \neg D^\rho).$$

The particular form of the substitutions θ_i guarantees the equivalence with the following

$$\neg\bigvee_{i=1}^n \exists \bar{Y}\,\neg D^{\rho\theta_i}\ .$$

6.5 □

In particular, Lemma 6.5 allows us to prove that the formula

$$\forall X_1X_2\,(Y \neq \{X_1, X_2\} \vee X_1 \doteq \emptyset)$$

(cf. example at the beginning of § 6.3.5) is equivalent to

$$\forall X_1X_2\,(Y \neq \{X_1, X_2\})\vee$$
$$\exists X_1X_2\,(Y \doteq \{X_1, X_2\} \wedge X_1 \doteq \emptyset \wedge X_2 \doteq \emptyset).$$

Using Lemma 6.5 instead of Lemma 6.4 (i.e, Chan's Lemma) allows us to effectively exploit constructive negation to implement intensional sets within the {log} language. The resulting version of this language, then, enables us to safely deal with a number of cases which, in contrast, cannot be dealt with in a satisfactory way using negation as failure.

In what follows we show a sample {log} program, i ntended to demonstrate such an enhancement of the expressive power of our language.

Example 6.6 *A combination of three or more tones sounded together in harmony is said to be a* chord. *There is an unambiguous symbolic notation, mainly used in rock and jazz scores, for denoting chords. We deal only with major and minor chords. It is easy to extend the program to any chord.*

The following numerical mapping for notes is assumed.

a	$\mapsto$	0	c♯, d♭	$\mapsto$	$s^4 0$	f	$\mapsto$	$s^8 0$
a♯, b♭	$\mapsto$	$s^1 0$	d	$\mapsto$	$s^5 0$	f♯, g♭	$\mapsto$	$s^9 0$
b	$\mapsto$	$s^2 0$	d♯, e♭	$\mapsto$	$s^6 0$	g	$\mapsto$	$s^{10} 0$
c	$\mapsto$	$s^3 0$	e	$\mapsto$	$s^7 0$	g♯, a♭	$\mapsto$	$s^{11} 0$

where, as usual, $s^n x$ *denotes the term* $\underbrace{s(\cdots(s}_{n}(x))\cdots)$. *Thus, a {log} program dealing with chords can be defined as follows:*

```
chord({T | R}, [T | S]) :−        %%% T for 'tonic'
    rest(R, S, T)
rest({X3, X5}, [min], T) :−       %%% min: minor chord
    interval(X3, T,$s^3$0),
    interval(X5, T,$s^7$0)
rest({X3, X5}, [ ], T) :−         %%% major chord
    interval(X3, T,$s^4$0),
    interval(X5, T,$s^7$0).
```

where the predicate interval *represents the minus operator (modulo* 12*) and is defined as follows:*

```
interval(X, Y, S) :−               minus(X, X, 0) :−
    minus(X, Y, S)                 minus($s$(X), Y,$s$(N)) :−
interval(X, Y, S) :−                   minus(X, Y, N).
    minus($s^{12}$X, Y, S)
```

Restricting our attention, for the sake of simplicity, to the first clause defining interval, *we can observe that, given a goal like*

```
:− chord(X, Y)
```

we get the following two answers

$X \doteq \{T, s^3T, s^7T\}, Y \doteq [T, min]$
$X \doteq \{T, s^4T, s^7T\}, Y \doteq [T]$

where, as usual, T *is intended to be implicitly existentially quantified. We can further observe that, thanks to the use of the enhanced version of constructive negation described in this section, {log} is able to answer correctly also to a goal of the form*

```
:− S ≐ {X : chord(X, Y) }
```

by providing the following solutions

$$\begin{array}{l}\mathsf{S} \doteq \emptyset, \forall \mathsf{T}\,(\mathsf{Y} \neq [\mathsf{T}]), \forall \mathsf{T}\,(\mathsf{Y} \neq [\mathsf{T}, \mathsf{min}]) \\ \mathsf{S} \doteq \{\,\{\mathsf{T}, s^3\mathsf{T}, s^7\mathsf{T}\}\,\}, \mathsf{Y} \doteq [\mathsf{T}, \mathsf{min}] \\ \mathsf{S} \doteq \{\,\{\mathsf{T}, s^4\mathsf{T}, s^7\mathsf{T}\}\,\}, \mathsf{Y} \doteq [\mathsf{T}].\end{array}$$

6.3.7 Undecidability Results

Lemma 6.5, which allows to extend the class of intensional sets handled using negation, requires that a number of hypothesis hold.

In this section we first show an example for which such conditions are not fulfilled; later we will present a strong undecidability result which points out the impossibility of cover all cases opened by the translation of intensional set formers.

Consider the goal

$$:\!-\;\; S \doteq \{\, Z : Z \in Y \,\}$$

which holds if and only if S and Y are the same set. According to the translation-based technique described in § 6.3 the following {log} program is generated from this goal:

$$\begin{array}{ll} \mathit{setof}_{\in}(S,Y) \leftarrow & \mathit{partial}_{\in}(S,Y) \leftarrow \\ \quad (\forall X \in S)(X \in Y), & \quad X \notin S, X \in Y. \\ \quad \neg\mathit{partial}_{\in}(S,Y) & \end{array}$$

A possible solution to the subgoal $\leftarrow\; (\forall X \in S)(X \in Y)$ is

$$S \doteq \{X\} \wedge Y \doteq \{X \,|\, N\}\,.$$

By solving $\leftarrow \mathit{partial}_{\in}(\{X\}, \{X \,|\, N\})$ then we get

$$\leftarrow X' \notin \{X\} \wedge X' \in \{X \,|\, N\}\,.$$

By applying sat to this constraint the only disjunct we get is

$$\exists X'\, N'\, (N \doteq \{X' \,|\, N'\} \wedge X' \neq X)\,.$$

The constructive negation approach performs the negation of such formula, leading to the global (partial) solution

$$S \doteq \{X\} \wedge Y \doteq \{X \,|\, N\} \wedge \forall X'\, N'\, (N \neq \{X' \,|\, N'\} \vee X' \doteq X)\,,$$

which is equivalent to

$$S \doteq \{X\} \wedge Y \doteq \{X \,|\, N\} \wedge \forall X' \, N' \, (N \doteq \{X' \,|\, N'\} \rightarrow X \doteq X') \, .$$

It states that X is the only element of Y, namely $Y \doteq \{X\}$. This is a sound answer.

Unfortunately we are unable to explode the universal quantification in order to reduce it to a normalized form. Applying the translation rule of the original constructive negation we obtain the solution:

$$S \doteq \{X\} \wedge Y \doteq \{X \,|\, N\} \wedge \forall X'N' \, (N \not\doteq \{X' \,|\, N'\}) \;\; \vee$$
$$S \doteq \{X\} \wedge Y \doteq \{X \,|\, N\} \wedge \exists X'N'(N \doteq \{X' \,|\, N'\} \wedge X \doteq X') \, .$$

The first disjunct implies that N does not contain elements. It is then equivalent to $S \doteq \{X\} \wedge Y \doteq \{X\}$.
The second one simplifies to (see redundant variable elimination in § 6.3.4) $S \doteq \{X\} \wedge Y \doteq \{X, X \,|\, N'\}$, i.e. $S \doteq \{X\} \wedge Y \doteq \{X \,|\, N'\}$. Any (even non-empty) set can be taken as N': a solution that it is wrong since it has been computed using Chan's Lemma, proved to be incorrect for {log}.

Moreover, any set unification algorithm applied to

$$\{X' \,|\, N'\} \quad \doteq \quad \{X'' \,|\, N''\}$$

reports the following most general unifiers:

(1) $[X''/X', N''/N']$
(2) $[X''/X', N''/\{X' \,|\, N'\}]$
(3) $[X''/X', N'/\{X'' \,|\, N''\}]$
(4) $[N'/\{X'' \,|\, M\}, N''/\{X' \,|\, M\}]$.

Only the first unifier satisfies the hypothesis of Lemma 6.5, hence the extended translation rule presented in such lemma is not applicable.

As mentioned, Vaught in [110] proves the *essential undecidability* of the theory NW. This means (see § 6.3.7) that any consistent extension of NW is undecidable. Hence, the undecidability of `WF_sets` follows, even when Σ is $\{\emptyset, \{\cdot \,|\, \cdot\}\}$.

Below we present a general technique to translate any formula written in the language $\Pi = \{ \doteq, \in \}$, $\Sigma = \{ \emptyset, \{\cdot \,|\, \cdot\} \}$ into a $CLP(\mathcal{S})$ program with negation.

Consider a prenex formula

$$\Psi = (\neg)\exists \bar{X}_1 \neg\exists \bar{X}_2 \neg\exists \bar{X}_3 \cdots \neg\exists \bar{X}_k \, (\psi_1 \vee \cdots \vee \psi_k)$$

```
procedure Buildprogram(Psi);
begin
    return("←"add_clauses($0$, Psi));
end

function add_clauses($i$, Psi);
begin
   case Psi of
      psi$_1 \vee \cdots \vee$ psi_$k$:          %% quantifier free formula
         for j := $1$ to $k$ do assert("$p_i$(varlist(Psi)) ← psi$_j$");
      ¬Phi:                             %% negated formula
         assert("$p_i$(varlist(Psi)) ← ¬" add_clauses($i+1$, Phi));
      $\exists\bar{X}$ Phi;                          %% existentially quantified formula
         assert("$p_i$(varlist(Psi)) ←" add_clauses($i+1$, Phi))
   end case;
   return("$p_i$(varlist(Psi))")
end
```

Figure 6.1: Rewriting a formula in $CLP(\mathcal{S})$

such that $FV(\psi_1, \ldots, \psi_k) = \{\bar{X}_1, \ldots, \bar{X}_k\}$. The procedure Buildprogram, defined in Fig. 6.1, applied to Φ, generates a recursion-free $CLP(\mathcal{S})$ program with negation P using less than $2 \cdot k$ new predicate symbols and returns a goal G.

The procedure Buildprogram is simply the top-level procedure which calls the recursive function add_clauses. The first parameter of the latter function is an integer number i which allows to produce clauses using newly generated predicate symbols. It uses the auxiliary functions assert, which updates a global set of clauses, and varlist, which returns the list of free variables of a formula.

For instance, the formula

$$\exists X_1 \neg\exists X_2 \neg\exists X_3 \, ((X_1 \in X_2 \wedge X_1 \doteq X_3) \vee (X_1 \in X_3))$$

will be translated in the following program P:

$$\begin{array}{rcl}
p_0 & \leftarrow & p_1(X_1) \\
p_1(X_1) & \leftarrow & \neg p_2(X_1) \\
p_2(X_1) & \leftarrow & p_3(X_1, X_2) \\
p_3(X_1, X_2) & \leftarrow & \neg p_4(X_1, X_2) \\
p_4(X_1, X_2) & \leftarrow & p_5(X_1, X_2, X_3) \\
p_5(X_1, X_2, X_3) & \leftarrow & X_1 \in X_2 \wedge X_1 \doteq X_3 \\
p_5(X_1, X_2, X_3) & \leftarrow & X_1 \in X_3
\end{array}$$

together with the goal

$$\leftarrow p_0$$

Since the program P obtained by the execution of Buildprogram is always recursion-free, its refutation tree is always finite; this means that any CLP implementing constructive negation will return a computed answer to G with respect to the program P if and only if the formula is satisfiable in the theory `wf_sets`.

Hence, such a CLP-intepreter would allow us to solve the satisfiability problem w.r.t. NW for quantified formulae, contradicting [110, 89, 16, 80].

Chapter 7

Programming in Logic with Sets

In this chapter we will present some examples of the high declarativeness style of Logic Programming with Sets. In § 7.1, devoted to well-founded sets, we will describe {log} programs solving a certain number of problems. In § 7.1.1 the power of set unification and of the constraint handling to prune the search tree is mainly used, while, in § 7.1.2, the programming style takes advantage of the expressive power of intensional set formers. In § 7.2 we will analyze the power of hyperset unification to solve, with a unique computation of the unification algorithm, the equivalence of automata problem, the *automata matching* problem, and the type finding problem.

Examples of § 7.1.1 can be found also in [36]; examples of § 7.1.2 can be found in [36] and [41]. The equivalence of automata problem using hypersets (a declarative—not computationally competitive—example) was firstly presented in [35].

7.1 Well founded sets

In this section we will use the CLP language {log}, presented in Chapter 6, to give examples of high declarative style logic programming.

7.1.1 Programming with extensional sets

Various standard set operations—e.g. union, intersection, difference, etc.—can be straightforwardly programmed in {log}; the {log} definition of these operations can be quite similar to the usual PROLOG definition obtainable by representing sets as lists (cf., e.g., [81]). For instance, the following three {log} clauses can be used to define the intersection of two sets:

```
∩({}, _, {}).
∩({X | A}, B, {X | C}) :-
    X ∉ A, X ∈ B, ∩(A, B, C).
∩({X | A}, B, C) :-
    X ∉ A, X ∉ B, ∩(A, B, C).
```

There are, however, a number of new facilities in {log}, especially devoted to set manipulation, which can make the behavior of {log} programs significantly better than that of the corresponding PROLOG programs and which can be exploited to drastically simplify the program development effort.

A first notable difference with respect to the PROLOG solution comes out when dealing with *non-ground sets.* Due to the ability to treat simple *set constraints*, {log} can provide answers to goals which are hardly managed in standard PROLOG. For example, given the {log} goal

```
:- ∩ ({X}, {Y}, Z).
```

we get from the above definition of the predicate intersection the following two answers (the second of which containing a negative constraint):

$$X \doteq Y, Z \doteq \{Y\}$$
$$X \neq Y, Z \doteq \{\}\,.$$

Another feature of {log} which strongly enhances the expressive power of the language, is the availability of *Restricted Universal Quantifiers.* Some definitions using such a facility—namely, the subset and disjoint predicates—have been shown in Chapter 6. Some further usages of RUQs will be shown in next examples.

A third important feature of {log} is *set-unification.* There are several problems—e.g., resource allocation problems and combinatorial problems in general—where the non-determinism embedded in the set-unification mechanism can be advantageously exploited to make programs simpler to write and more declarative to understand than those obtainable by using conventional PROLOG programming techniques.

We prove this by showing the {log} solutions to two well-known combinatorial problems, namely the SEND + MORE = MONEY puzzle and the coloring of a map.

Cryptarithmetic puzzle. This is the well-known problem of solving the equation *SEND + MORE = MONEY* by assigning a distinct digit between 0 and 9 to each letter appearing in it. A classical solution to this problem can be developed by adopting a *generate & test* approach, i.e., by successively generating assignments of digits to the letters of the puzzle and then testing, for each assignment, whether the generated pattern satisfies the required constraints.

By using sets and the extended resolution procedure (in particular, set unification), one can avoid the explicit use of a generator and of mechanisms for backtracking in search for new solutions.

```
solve_puzzle(S, E, N, D, M, O, R, Y) :−
    {S, E, N, D, M, O, R, Y, _, _} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9},
     M ∗ 10000   +   O ∗ 1000   +  N ∗ 100   +  E ∗ 10   +  Y
                =:=  S ∗ 1000   +  E ∗ 100   +  N ∗ 10   +  D
                 +   M ∗ 1000   +  O ∗ 100   +  R ∗ 10   +  E .
```

Clearly, it would be nice to replace the extra-logical literal involving =:= by an integer equation (or by a system of integer equations and inequalities), assuming that a suitable integer constraint solver interacts with the set unifier. As usual, a drastic reduction of the search space can in fact ensue from the interplay between two solvers.

Coloring. Given a set $\{c_1, \ldots, c_m\}$ of Colors, a set $\{R_1, \ldots, R_n\}$ of Regions, $m \leq n$, and a set Map $= \{\{R_{i_1}, R_{j_1}\}, \ldots, \{R_{i_k}, R_{j_k}\}\}$, $i_p \neq j_p$ for all $p = 1, \ldots, k$, of pairs of neighboring regions, the predicate coloring returns an assignment of colors to the regions so that no two neighboring regions have the same color and all colors are used.

$$\begin{array}{l}\mathsf{coloring}(\mathsf{Regions}, \mathsf{Map}, \mathsf{Colors}) \text{ :- } \\ \quad \mathsf{Regions} \doteq \mathsf{Colors}, \\ \quad (\forall \mathsf{R} \in \mathsf{Regions})(\{\mathsf{R}\} \notin \mathsf{Map}).\end{array}$$

A sample goal:

$$\text{:- } \mathsf{coloring}(\{\mathsf{R1}, \mathsf{R2}, \mathsf{R3}\}, \{\{\mathsf{R1}, \mathsf{R2}\}, \{\mathsf{R2}, \mathsf{R3}\}\}, \{\mathsf{c1}, \mathsf{c2}\}).$$

returns the constraint answers

$$\begin{array}{l}\mathsf{R1} \doteq \mathsf{c1} \wedge \mathsf{R2} \doteq \mathsf{c2} \wedge \mathsf{R3} \doteq \mathsf{c1} \\ \mathsf{R1} \doteq \mathsf{c2} \wedge \mathsf{R2} \doteq \mathsf{c1} \wedge \mathsf{R3} \doteq \mathsf{c2} .\end{array}$$

As mentioned in Chapter 6, {log} has been developed as a general-purpose programming language. Nevertheless, there seem to be certain application areas in which the use of sets fits more naturally, allowing some definite improvements both in the quality and in the development time of the final software product. These ‘interesting’ areas include database applications (see for instance [66, 83]), combinatorial problems, graph-related applications and operational research in general (e.g., resource allocation problems), as pointed out for instance in [46, 68].

To conclude, we show also a simple example in a quite unusual application area, namely music writing, where sets turn out to be a very natural notation to represent collections of notes.

Music. We give here an alternative definition of the predicate chord, presented in Example 6.6. A combination of three or more tones sounded together in harmony is said to be a *chord.* There is an unambiguous symbolic notation, mainly used in rock and jazz scores, for denoting chords.

Writing a program that, given the symbolic denotation, returns the corresponding chord, is a simple task in any programming language. Solving the reverse problem is more difficult, since chords are genuine sets and each string of symbols usable to denote a chord is based on only one of the notes in it (the *tonic* note). Both problems are solved by the following {log} program, working for 3-notes chords. It is easy to extend it so as to treat n-notes chords (in jazz music 7-notes chords are common).

Note representation: for the sake of simplicity we assume a unique representative for each altered note (e.g. b♭ stands also for a♯).

```
note(a, 0).    note(b♭, 1).  note(b, 2).    note(c, 3).
note(d♭, 4).   note(d, 5).   note(e♭, 6).   note(e, 7).
note(f, 8).    note(g♭, 9).  note(g, 10).   note(a♭, 11).
```

The {log} program

```
chord({T | R}, [T | S]) :-        %%% T for 'tonic'
    rest(R, S, T).
rest({X3, X5}, [min], T) :-       %%% min: minor chord
    interval(X3, T, 3),interval(X5, T, 7).
rest({X3, X5}, [], T) :-          %%% major chord
    interval(X3, T, 4),interval(X5, T, 7).
rest({X3, X5}, [maj5], T) :-      %%% maj5: augmented chord
    interval(X3, T, 4),interval(X5, T, 8).
rest({X3, X5}, [dim], T) :-       %%% dim: diminished chord
    interval(X3, T, 3),interval(X5, T, 6).
interval(Y, X, Offset) :-         %%% offset determination
    note(X, N1),N2 is (N1 + Offset) mod 12,note(Y, N2).
```

Figure 7.1: A music example

Figure 7.1 reports the program; sample goals are the following:

$:-\ \mathsf{chord}(\{\mathsf{g}, \mathsf{e}\flat, \mathsf{g}, \mathsf{b}\flat\}, \mathsf{X}).$
$\quad \mathsf{X} \leftarrow [\mathsf{e}\flat]$
$:-\ \mathsf{chord}(\mathsf{X}, \mathsf{Y}).$
$\quad \mathsf{X} \leftarrow \{\mathsf{a}, \mathsf{c}, \mathsf{e}\}, \mathsf{Y} \leftarrow [\mathsf{a}, \mathsf{min}]$
$\quad\quad \vdots$
$\quad \mathsf{X} \leftarrow \{\mathsf{a}\flat, \mathsf{b}, \mathsf{d}\}, \mathsf{Y} \leftarrow [\mathsf{a}\flat, \mathsf{dim}].$

$:-\ \mathsf{chord}(\mathsf{X}, [\mathsf{f}, \mathsf{min}]).$
$\quad \mathsf{X} \leftarrow \{\mathsf{f}, \mathsf{a}\flat, \mathsf{c}\}$

7.1.2 Programming with intensional sets

Let us see a few simple examples aimed at showing the *declarative programming style* supported by intensional sets.

As a first example, we show a very straightforward definition of the intersection predicate (see § 7.1) using intensional sets:

$\cap(\mathsf{A}, \mathsf{B}, \{\mathsf{X} : \mathsf{X} \in \mathsf{A}, \mathsf{X} \in \mathsf{B}\}).$

Other basic set-theoretic operations can be redefined using intensional sets in a very similar way.

Prime numbers. Our second example is the definition of a predicate, primes, that computes all the prime numbers smaller than a given N:

$\mathsf{primes}(\mathsf{N}, \{\mathsf{X} : \mathsf{between}(1, \mathsf{N}, \mathsf{X}),\ \mathsf{is\_prime}(\mathsf{X})\}).$
$\mathsf{is\_prime}(\mathsf{X})\ :-$
$\quad (\forall \mathsf{Z} \in \{\mathsf{Y}\ :\ \mathsf{between}(1, \mathsf{X}, \mathsf{Y})\})\mathsf{non\_div}(\mathsf{Z}, \mathsf{X}).$
$\mathsf{between}(\mathsf{A}, \mathsf{B}, \mathsf{C})\ :-\ \mathsf{A} < \mathsf{C}, \mathsf{C} < \mathsf{B}.$
$\mathsf{non\_div}(\mathsf{A}, \mathsf{B})\ :-\ 0 =\backslash= (\mathsf{B}\ \mathtt{mod}\ \mathsf{A}).$

Connected components of a graph. The next example regards undirected graphs. Assuming that every node belongs to at least one arc, a set Arcs of doubletons suffices to represent a graph.

The set of all nodes of a given graph can be determined as follows:

$\mathsf{nodes}(\mathsf{Arcs}, \{\mathsf{X} : \{\mathsf{X}, \_\} \in \mathsf{Arcs}\}).$

To determine the set of all connected components of a given graph Arcs of cardinality N, we can exploit the following predicate:

```
components(1, Arcs, {Z : Z =. {X}, nodes(Arcs, Nodes), Z ∈ Nodes}).
components(N, Arcs, Comp) :−
   N > 1,
   N1 is N − 1, components(N1, Arcs, Comp1),
   Comp = {Z : Z =. {X | C}, Y ∈ C, C ∈ Comp1, X ∉ C, {X, Y} ∈ Arcs}.
```

Finite State Automata. Let us consider an *NFSA* $M = \langle \Sigma, Q, q_0, F, \delta \rangle$, whose transition function $\delta : Q \times \Sigma \rightarrow Pow(Q)$ we assume to have been modeled by the relation

$$\mathsf{A} = \{[q, s, \delta(q, s)] \ : \ q \in Q, s \in \Sigma\} \, .$$

We define the predicate delta_star that performs the transitive closure of a given transition function δ on a given input string S.

```
delta_star(SetQ, A, [ ], SetQ).
delta_star(SetQ_in, A, [S | R], SetQ_out) :−
     delta_star({Q : step(SetQ_in, A, S, Q)}, A, R, SetQ_out).
step(SetQ, A, S, Q) :−
     Q_p ∈ SetQ, [Q_p, S, SetQ_p] ∈ A, Q ∈ SetQ_p.
```

A predicate that checks whether or not a string S is accepted by an automaton defined by its transition function A, its initial state q_0, and its set Final of final states can be defined as follows:

```
accepted(S, A, Final) :−
     delta_star({q0}, A, S, States), Z ∈ States, Z ∈ Final.
```

It is now straightforward to define procedures capable of testing common properties of FSA. For example, the following fragment of {log} program is intended to verify the equivalence between two deterministic FSA (it is just a direct application of the classical pumping lemma for DFSA, and we are not concerned here with the efficiency of the resulting code):

$$
\begin{array}{l}
\mathsf{equivalent}(\mathsf{A}_1, \mathsf{Final}_1, \mathsf{A}_2, \mathsf{Final}_2, \mathsf{Sigma}) :- \\
\quad \mathsf{states}(\mathsf{A}_1, \mathsf{N}_1), \mathsf{states}(\mathsf{A}_2, \mathsf{N}_2), \mathsf{maximum}(\mathsf{N}_1, \mathsf{N}_2, \mathsf{N}), \\
\quad \mathsf{S} \doteq \{\mathsf{X} : \mathsf{string}(\mathsf{Sigma}, \mathsf{N}, \mathsf{X})\}, \\
\quad \{\mathsf{X} : \mathsf{X} \in \mathsf{S}, \mathsf{accepted}(\mathsf{X}, \mathsf{A}_1, \mathsf{Final}_1)\} = \\
\quad \{\mathsf{X} : \mathsf{X} \in \mathsf{S}, \mathsf{accepted}(\mathsf{X}, \mathsf{A}_2, \mathsf{Final}_2)\}.
\end{array}
$$

where

- $\mathsf{states}(\mathsf{A}, \mathsf{N})$ holds if N is the number of states of the automaton A;
- $\mathsf{maximum}(\mathsf{N}_1, \mathsf{N}_2, \mathsf{N})$ states that N is the maximum between N_1 and N_2;
- $\mathsf{string}(\mathsf{Sigma}, \mathsf{N}, \mathsf{X})$ holds if X is a string on the alphabet Sigma of length less than or equal to N.

Abstract Interpretation An *Abstract Interpreter* is a static analysis tool used to extract information on properties of the program to be analyzed. Its main goal is to rewrite the original program substituting the actual operations (which implicitly work on the Herbrand universe) with their *abstract* equivalents on the abstract domain. In particular, the various operations performed during standard resolution (e.g., unification, application of substitutions) need to be properly abstracted. In the *Ms system* [32], for instance, each clause for a predicate *p* in the original program is rewritten according to a suitable translation scheme and stored with a modified head p\$cl. Moreover, an additional clause with head p\$pred is introduced as a driver for the execution of a *p-based* goal, with the task of allowing the test of all the different possible paths (i.e., all the matching clauses) explicitly, without using a failure-driven loop. The result of the analysis of the different paths is obtained by taking the *least upper bound* of the results obtained from every individual path—i.e. the value of the studied property for a predicate is obtained by taking the 'upper bound' among the values obtained from the different clauses defining the predicate.

In the *Ms* system the explicit iteration over the different clauses defining a predicate requires a considerably complex mechanism.

The same process can be described in a one-line {log} clause using intensional sets:

$$\begin{array}{l}\mathsf{p\$pred(InMode, OutMode)} \leftarrow \\ \qquad \mathsf{lub}(\{\mathsf{Out} \mid \mathsf{p\$cl(I, InMode, Out)} \wedge 1 \leq \mathsf{I} \leq \mathsf{m}\}, \mathsf{OutMode}).\end{array}$$

where I is the index of the considered clause, sfm is the number of clauses defining p, InMode is the input value for the considered property (e.g., the set of modes previously computed), and OutMode will be instantiated to the output value for the considered property.

Considering that the description of the property is extracted from the source program itself, and as such it can contain variables, the collection of the intensional set in the p$pred clause requires the use of constructive negation (i.e., it does not belong to the cases covered by the use of negation as failure).

7.2 Non-well-founded sets

One of the most common exploitations of hypersets is as a means to model deterministic finite state automata (cf., e.g., [14]). As we will now see, the hyperset universe and unification algorithms are sufficiently powerful to offer algorithmic support to such modeling task.

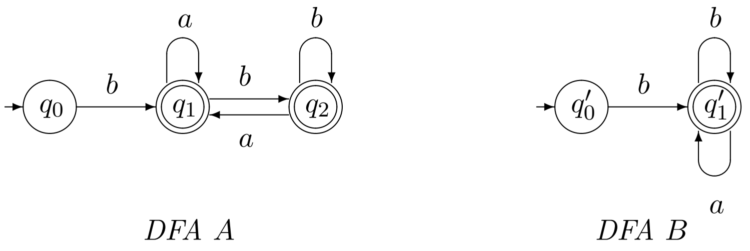
A *deterministic finite automaton* (DFA for short) consists of a set $Q = \{Q_0, \ldots, Q_n\}$ of *states*, a set $S = \{s_1, \ldots, s_k\}$ of *symbols*, and a *transition function* $d : Q \times S \to Q \cup \{\perp\}$. One of the states—say Q_0—is called *initial state*, and there is a set $F \subseteq Q$ of *accepting states* (for a complete definition of DFAs see, for instance, [52]).

Given a DFA A, one may define a corresponding Herbrand system $\mathcal{E}^A$ in the signature $\Sigma = \{\emptyset, \{\cdot \mid \cdot\}, \perp, \ \delta, \delta'\}$, where $\perp$ is a constant symbol and δ, δ' are functional symbols of arity k, as follows:

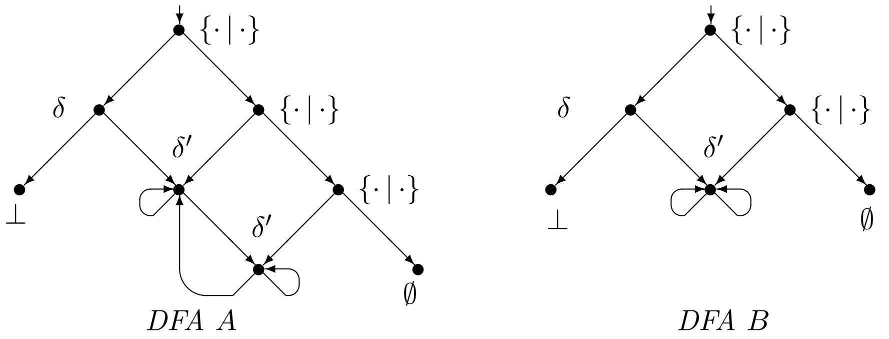
$$\begin{array}{rcl}\mathcal{E}^A & = & A \doteq \{Q_0, \ldots, Q_n\} \wedge \\ & & \bigwedge_{Q_i \text{ in } Q \setminus F} Q_i \doteq \delta(d(Q_i, s_1), \ldots, d(Q_i, s_k)) \wedge \\ & & \bigwedge_{Q_i \text{ in } F} Q_i \doteq \delta'(d(Q_i, s_1), \ldots, d(Q_i, s_k))\end{array}$$

This can easily be re-expressed as an equivalent flat system, or, if one prefers, as a graph bearing the same information. Other techniques to simulate DFAs by hypersets have been proposed in the literature. For example, in [14], one such technique is presented, and it is shown that the notion of bisimulability between graphs $\approx$ (cf. Def. 3.24) corresponds exactly to equivalence between automata.

Given two DFAs A and B, it is easy to determine whether or not they accept the same language, as is shown by the following simple example. Consider the two DFAs



where the sets of accepting states are $F_A = \{q_1, q_2\}$ and $F_B = \{q'_1\}$, respectively. In our framework these automata can be modeled by the following graphs:



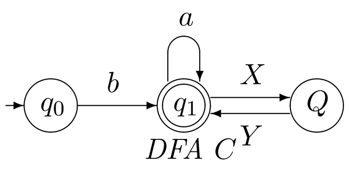
or by means of the two systems

$$\begin{array}{rcl}
\mathcal{E}^A & = & A \doteq \{Q_0 \,|\, X_1\} \wedge X_1 \doteq \{Q_1 \,|\, X_2\} \wedge X_2 \doteq \{Q_2 \,|\, X_3\} \wedge X_3 \doteq \emptyset \wedge \\
 & & Q_0 \doteq \delta(\perp, Q_1) \wedge Q_1 \doteq \delta'(Q_1, Q_2) \wedge Q_2 \doteq \delta'(Q_1, Q_2)\,, \\
\mathcal{E}^B & = & B \doteq \{Q'_0 \,|\, Y_1\} \wedge Y_1 \doteq \{Q'_1 \,|\, Y_2\} \wedge Y_2 \doteq \emptyset \wedge \\
 & & Q'_0 \doteq \delta(\perp, Q'_1) \wedge Q'_1 \doteq \delta'(Q'_1, Q'_1)\,.
\end{array}$$

The function call $\mathsf{Unify}(\mathcal{E}^A \wedge \mathcal{E}^B \wedge A \doteq B \wedge Q_0 \doteq Q'_0)$ has (at least) one satisfactory branch, reporting the system in solvable form $Q_1 \doteq Q_2 \wedge Q'_1 \doteq Q_2 \wedge \ldots$; hence the two automata are unifiable (bisimulation $\approx$ holds between their graphs).

7.2.1 Automaton matching

The problem we have considered in the above example could have been solved with more efficient algorithms (cf. [4]). However, notice that the actual question we can answer using a unification algorithm for hybrid hypersets, is that of determining whether two *partially defined* DFAs can be completed in the same automata. Hence, the former problem stands to the latter as ground comparison stands to matching. As an example, one may be interested if the following partially defined automata,



DFA C

where X, Y, and Q are unknowns, can be completed so as to accept the same language as above defined *DFA B*. This problem is in fact NP-complete (cf. § 4.1); in order to prove its NP-hardness, consider the following reduction. Given the formula (instance of 3-SAT)

$$\Phi = (\ell_1^1 \vee \ell_2^1 \vee \ell_3^1) \wedge \cdots \wedge (\ell_1^m \vee \ell_2^m \vee \ell_3^m)$$

with

$\ell_i^j = a_i^j$ or $\ell_i^j = \neg a_i^j$, where
$a_i^j \in \{d_1, \ldots, d_k\}, \forall i \in \{1, 2, 3\}, \forall j \in \{1, \ldots, m\}, k \leq 3 \cdot m,$

we introduce distinct variables $X_1, Y_1, \ldots, X_k, Y_k$, and define the transformation function f as follows:

$$f(\ell) = \begin{cases} X_i & \text{if } \ell \equiv d_i, \\ Y_i & \text{if } \ell \equiv \neg d_i, \end{cases}$$

Thus, for solving the instance Φ of 3-SAT, it is sufficient to check whether there exists a substitution for the variables in the automaton

$$\begin{tikzcd}
q_0 \arrow[r, "X_1|Y_1"] & q_1 \arrow[r, "X_2|Y_2"] & \cdots \arrow[r, "X_k|Y_k"] & q_k \arrow[r, "0|f(\ell_1^1)", "f(\ell_2^1)|f(\ell_3^1)"'] & \cdots \arrow[r, "0|f(\ell_1^m)", "f(\ell_2^m)|f(\ell_3^m)"'] & q_{k+m}
\end{tikzcd}$$

that makes it equivalent to the fully specified automaton below:

$$\begin{tikzcd}
q_0 \arrow[r, "0|1"] & q_1 \arrow[r, "0|1"] & \cdots \arrow[r, "0|1"] & q_k \arrow[r, "0|1"] & \cdots \arrow[r, "0|1"] & q_{k+m}
\end{tikzcd}$$

NP-completeness ensues from the encoding of the automata completion problem into the hyperset unification one (presented above), proved to be NP-complete in § 4.1.

7.2.2 Type finding

This section is devoted to describe how non-well-founded set unification is suitable for type checking and type finding. Adapting the definition in Chapter 6 of [5], an example of *type expression* can be inductively defined as follows:

1. a basic type (boolean, char, integer, real) is a type expression;
2. a (type) variable is a type expression;
3. if t is a type expression and n_1, n_2 are natural numbers, then $\mathsf{array}(\langle n_1,n_2\rangle,t)$ is a type expression;
4. if $t^1_1, \ldots, t^1_{k_1}, \ldots, t^n_1, \ldots, t^n_{k_n}$ are type expressions and $i_1, \ldots, i_n$ are field names (i.e. identifiers), then
   $$\mathsf{record}(\{\langle i_1, \{t^1_1, \ldots, t^1_{k_1}\}\rangle, \ldots, \langle i_n, \{t^n_1, \ldots, t^n_{k_n}\}\rangle\})$$
   is a type expression;
5. if t is a type expression, then $\mathsf{pointer}(t)$ is a type expression.

Most Pascal type declarations uniquely define type expressions. For instance,

$$\mathsf{vector} = \mathsf{array}\ [1..10]\ \mathsf{of\ char}$$

defines the type expression

$$\mathsf{array}(\langle 1, 10\rangle, \mathsf{char})\ .$$

Furthermore, in order to infer data types from a program without type declaration, also *non ground* (i.e. partially specified) data types are allowed to be type expressions.

Notice that the data structure '$\{\cdots\}$' (set) used as argument in record captures the semantics of the record data type better than the list one, since ordering of fields is immaterial. For instance, the PASCAL definition

```
t_one  =  record
               next : list;
               data : char
          end
```

defines the same data types as

```
t_two  =  record
               data : char;
               next : list
          end
```

Moreover, *sets* was conveniently used to introduce multiple valued fields, namely the *union* data type.

When data types are recursively defined, a rational infinite tree representation is needed. In this case, the Herbrand system representation is, clearly, more suitable.

A *type constraint* is a system of equations between type expressions. A *type system* is a collection of rules for assigning type constraints to the various parts of a program.

Fig. 7.2 illustrates how a type system works. For the sake of simplicity, assume that different procedures/functions use different variable names (such problem can be overcome via renaming). We will make use of the following assumptions:

- next is a keyword with the meaning: the data type of $\langle \mathsf{id} \rangle \uparrow$ .next is the same as $\langle \mathsf{id} \rangle$, and
- the data type of the keyword nil matches with any pointer data type.

PASCAL-like program		Type constraint
new(p);	$\Rightarrow$	$P \doteq \mathsf{pointer}(P_1)$,
p $\uparrow$ .next := nil;	$\Rightarrow$	$P_1 \doteq \mathsf{record}(\{\langle \mathsf{next},\{P\}\rangle \mid R_1\})$,
p $\uparrow$ .mode := true;	$\Rightarrow$	$P_1 \doteq \mathsf{record}(\{\langle \mathsf{mode}, \{\mathsf{boolean} \mid C_1\}\rangle \mid R_2\})$,
p $\uparrow$ .data := 1965;	$\Rightarrow$	$P_1 \doteq \mathsf{record}(\{\langle \mathsf{data}, \{\mathsf{integer} \mid C_2\}\rangle \mid R_3\})$,
new(q);	$\Rightarrow$	$Q \doteq \mathsf{pointer}(Q_1)$,
q $\uparrow$ .mode := false;	$\Rightarrow$	$Q_1 \doteq \mathsf{record}(\{\langle \mathsf{mode}, \{\mathsf{integer} \mid D_1\}\rangle \mid S_1\})$,
q $\uparrow$ .data := p;	$\Rightarrow$	$Q_1 \doteq \mathsf{record}(\{\langle \mathsf{data}, \{P \mid D_2\}\rangle \mid S_2\})$,
q $\uparrow$ .next := p;	$\Rightarrow$	$Q_1 \doteq \mathsf{record}(\{\langle \mathsf{next},\{Q\}\rangle \mid S_3\}),\ P \doteq Q$

Figure 7.2: A type system action

$$
\begin{aligned}
P \doteq \mathsf{pointer}(\,\mathsf{record}(\{ \\
&\langle \mathsf{next}, \{P\}\rangle, \\
&\langle \mathsf{mode}, \{\mathsf{boolean} \mid C_1\}\rangle, \\
&\langle \mathsf{data}, \{\mathsf{integer}, P \mid C_4\}\rangle, \\
\mid R\}))
\end{aligned}
$$

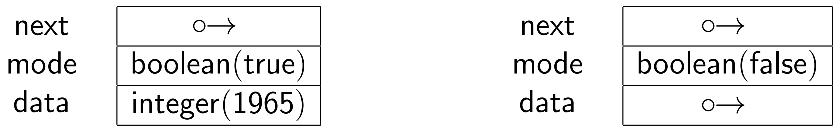


Figure 7.3: A solution to the type constraint

P and Q represent the type of p and of q, respectively.

One of the possible solvable form systems returned by the non-well-founded set unification algorithm over the type constraint above is represented in Fig. 7.3. Observe that recursive types are modeled by circular memberships.

Appendix A

The first part of the Appendix (§ A.1–A.3) deepens marginal aspects of the theoretical foundations of the aggregate theories analyzed in this thesis.

§ A.4, drawn from [36], shows how restricted universal quantifiers are a purely syntactical extension of the language {log}.

In § A.5, presented in [34], it is shown how to rewrite an hybrid (well-founded or not) set unification problem into a pure set unification one. § A.6, appendix also of [11], reports the number of independent unifiers that a minimal unification algorithm (cf. § 4.4) should return to given set unification problems.

A.1 Flatland

All aggregate theories presented in the thesis are, in a sense, two-dimensional. Namely, a list (bag, compact list, set) can grow either in length (number of elements) or in depth (nesting). In this section we will presented minimal theories of set using objects that develops in one direction (§ A.1.1) or in no direction (§ A.1.2).

A.1.1 Mono-dimensional theories

First a brief remark about set theories in which sets can grow only in length (flat sets): they have sense only in hybrid contexts, in which only ur-elements (cf. Def. 3.35) can take the role of elements of sets. In other words, we need to add an axiom of the form:

$$\forall x \left((\exists y \in x) \to (\forall y \in x)(\forall z\,(z \notin y))\right)$$

to the theory `WF_sets`, which ensures that no nesting of sets is allowed.

In the rest of the section we present a theory of sets which can grow only in depth; as a starting point we consider the *standard structure* for $\langle \omega, 0, S\rangle$. Γ_{0S} is the (complete) theory identified by the signature $\Pi = \{\doteq\}$ and $\Sigma = \{0, S\}$, where $ar(0) = 0$ and $ar(S) = 1$, and by the recursive set of axioms

$$\begin{array}{ll}
(1) & \forall x\, y \left(S(x) \doteq S(y) \to x \doteq y\right) \\
(2) & \forall x \left(0 \not\doteq S(x)\right) \\
(3.1) & \forall x \left(x \not\doteq S(x)\right) \\
(3.2) & \forall x \left(x \not\doteq S(S(x))\right) \\
(3.3) & \forall x \left(x \not\doteq S(S(S(x)))\right) \\
\vdots & \vdots \\
(4) & \forall x \left(x \doteq 0 \vee \exists y (x \doteq S(y))\right)
\end{array}$$

Note that axioms (1) and (2) are exactly freeness axiom (F_1) and (F_2), respectively. Axioms $(3.1), (3.2), (3.3), \ldots$ represent the infinite expansion of axiom schema (F_3) (cf. § 3.2). Axiom (4) is exactly (DCA). We will use the notation $S^n t$ for $\underbrace{S(\cdots(S}_{n}(t))\cdots)$ (in particular, $S^0(t) = t$). For instance, (F_3) can be written as *for any positive integer* n, $\forall x\,(x \not\doteq S^n x)$.

Let $M = \langle D, I\rangle$ be a model of Γ_{0S}. Then

- D should provide a distinct element (numeral) $\underline{0}, \underline{1}, \underline{2}, \underline{3}, \ldots$ for all the terms

  $$S^0 0, S^1 0, S^2 0, S^3 0, \ldots;$$

- assume $d \in D$ be such that $d \neq S^0 0, d \neq S^1 0, d \neq S^2 0, d \neq S^3 0, \ldots$. By (DCA), exist y_1 such that $d \doteq S(y_1)$, and y_2 such that $y_1 \doteq S(y_2)$, and so on. We refer to such y_i as to $S^{-i} d$ (hence $d = S^n S^{-n} d$). Clearly, also an element witnessing $S^n d$ should belong to D, for any n. It is easy to see that $S^i d$ is different from $S^j 0$ for any i and j.

Note that axiom scheme (F_3) forces any pairs of elements in the same chain to be distinct.

Any domain D contains always an isomorphic copy of ω and, possibly, a number of isomorphical copies of the set of integer numbers $\mathbb{Z}$:

$$\begin{array}{ccccccccccccccc}
 & & & & & & & \circ & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \cdots \\
\cdots & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \cdots \\
\cdots & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \xleftarrow{S} & \bullet & \cdots \\
 & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots &
\end{array}$$

The particular form of the models of Γ_{0S} ensures that two models having the same uncountable infinite cardinality are isomorphical. Hence, thanks to the test of Łoś-Vaught (cf. [27]), Γ_{0S} is complete. However, a direct proof of this fact can be given by describing a quantifier elimination procedure (cf. [44]).[1] The only thing to prove is

Theorem A.1 *Let $\mathcal{L} = \langle \Pi, \Sigma \rangle$; then for any formula $\exists x\,(\ell_0 \wedge \cdots \wedge \ell_n)$, where ℓ_i's are literals, exists an open formula ψ such that*

$$\Gamma_{0S} \ \vdash \ \exists x\,(\ell_0 \wedge \cdots \wedge \ell_n) \leftrightarrow \psi\,.$$

Proof. We describe an algorithm that, accepting a conjunction $\ell_0 \wedge \cdots \wedge \ell_n$ as input, returns an open formula ψ as output. The empty conjunction is viewed as the generic tautology (i.e. $0 \doteq 0$).

1. removing of tautologies:
$$\begin{array}{lllllll}
S^m x & \doteq & S^m x & \wedge & C & \mapsto & C \\
S^m x & \not\doteq & S^n x & \wedge & C & \mapsto & C \\
S^m x & \doteq & S^n x & \wedge & C & \mapsto & \texttt{false} \\
S^m x & \not\doteq & S^m x & \wedge & C & \mapsto & \texttt{false}
\end{array}$$

2. x goes on the left:
$$\begin{array}{lllllll}
t & \doteq & S^m x & \wedge & C & \mapsto & S^m x \doteq t \wedge C \\
t & \not\doteq & S^m x & \wedge & C & \mapsto & S^m x \not\doteq t \wedge C
\end{array}$$

[1] A similar result concerning with a stronger theory can be found in [94].

3. substitution application (based on the fact $s \doteq t \leftrightarrow S^m s \doteq S^m t$):
   $S^m x \ \doteq \ t \ \wedge \ C \ \mapsto \ t \not\doteq 0 \wedge \cdots \wedge t \not\doteq S^{m-1}0 \wedge C'$
   where C' is obtained from C replacing
   $S^k x \ \doteq \ t'$ with $S^k t \ \doteq \ S^m t'$
   $S^k x \ \not\doteq \ t'$ with $S^k t \ \not\doteq \ S^m t'$

4. elimination of x (if no substitution has been applied):
   $S^m x \ \not\doteq \ t \ \wedge \ C \ \mapsto \ C$

The correctness of the algorithm is easy to prove.

A.1 $\square$

As example of application of the technique described in the proof of theorem A.1, let us analyze axiom *(DCA)*, i.e. $\forall y \left(y \doteq 0 \vee \exists x(y \doteq Sx)\right)$. The disjunct $\exists x(y \doteq Sx)$ becomes (action 2) $Sx \doteq y$, (action 3) $y \not\doteq 0$. The complete formula becomes $\forall y\,(y \doteq 0 \vee y \not\doteq 0)$, namely, a tautology.

We extend the language described by introducing the membership relation symbol $\in$, whose behavior is controlled by the axiom

$$(\in) \qquad x \in y \leftrightarrow y \doteq S(x)\ .$$

It is easy to see that the theory remains complete and, for instance, that the extensionality axiom

$$(E) \qquad x \doteq y \leftrightarrow \forall z\,(z \in x \leftrightarrow z \in y)$$

is derivable from $\Gamma_{0S}(\in)$. This can be proved using the quantifier elimination technique described in the proof of theorem A.1.

In the rest of this section we will illustrate as such theory can be seen as a nested (set) theory of singletons. The first two axioms are

$$\begin{array}{ll} (N) & \exists z\, \forall x\,(x \notin z) \\ (S) & \forall y \exists z \left(y \in z \wedge (\forall x \in z)(x \doteq y)\right), \end{array}$$

that can be skolemized into

$$\begin{array}{ll} (N) & \forall x\,(x \notin \emptyset) \\ (S_1) & x \in \{x\} \\ (S_2) & x \in \{y\} \rightarrow x \doteq y\ . \end{array}$$

The theory NS_1S_2 is strong enough to prove lemma 3.1; this means, in particular, that axiom (1) above holds for NS_1S_2.

Let us define the predicate *singleton* as follows

$$\mathit{singleton}(x) \quad \text{if and only if} \quad \exists y\Big(y \in x \wedge (\forall z \in x)(z \doteq y)\Big)\,.$$

It is easy to prove the following weak form of extensionality

$$\begin{aligned} NS_1S_2 \;\vdash\; & \mathit{singleton}(x) \wedge \mathit{singleton}(y) \rightarrow \\ & \Big(x \doteq y \leftrightarrow \forall z\,(z \in x \leftrightarrow z \in y)\Big)\,. \end{aligned}$$

However, the existence of non-singleton sets is consistent for NS_1S_2. In order to re-state S_0 as a set theory, we need to introduce another axiom:

$$(S_3) \qquad \forall x\Big(\exists y\,(y \in x) \rightarrow \mathit{singleton}(x)\Big)$$

In this case we can prove

$$NS_1S_2S_3 \;\vdash\; x \doteq y \leftrightarrow \forall z\,(z \in x \leftrightarrow z \in y)$$

or, in other words

$$NS_1S_2S_3 \;\vdash\; \{x\} \doteq \{y\} \rightarrow x \doteq y\,,$$

which is the counterpart of axiom (2) above.

Until nothing has been stated about well-foundedness of the singletons of the theory, a non-well-founded model of $NS_1S_2S_3$ in which, in particular, $\Omega = \{\Omega\}$ can be described (see [3]). This means that axiom (F_3) (and hence axiom schema (φ)) can be falsified. The regularity (foundation) axiom

$$(R) \qquad \exists z\,\forall y\,\Big(y \in x \rightarrow (z \in x \wedge y \notin z)\Big)\,,$$

ensures, in particular, that an element d such that $d = \{d\}$ cannot exists. However, because of the singleton axiom S_3, it cannot exclude the existence of a circle of membership of the form $x_1 \in x_2, x_2 \in x_3, \ldots x_{n-1} \in x_n, x_n \in x_1$, if $n > 1$. We need then to introduce the counterpart of axiom schema (F_3): for any positive integer n

$$(F_3) \qquad x \neq \underbrace{\{\cdots\{}_{n} x\}\cdots\}\,.$$

An isomorphism between $NS_1S_2S_3F_3$ and $F_1F_2F_3 \in (DCA)$ is evident.

A.1.2 Zero-dimensional theories

In this section we present a *zero-dimensional* theory. Only one individual $\bar{k}$[2] belongs to the interpretation domain D of any model $\langle D, I\rangle$. Let, as usual, $\Pi = \{\doteq, \in\}$; two interpretation are possible:

$$\begin{array}{llllll} (i) & I_i & \models & \bar{k} \doteq \bar{k} & I_i & \models \; \bar{k} \notin \bar{k} \\ (ii) & I_{ii} & \models & \bar{k} \doteq \bar{k} & I_{ii} & \models \; \bar{k} \in \bar{k}\,. \end{array}$$

(i) is a well-founded interpretation, while (ii) is non-well-founded.

The (non skolemized) theory which has $\langle\{k\}, I_i\rangle$ as unique model (modulo isomorphism) is the following:

$$(K) \qquad \exists x \left(x \doteq x \wedge \forall y\,(y \notin x) \wedge \forall y\,(y \doteq x)\right);$$

in other words, when $\Sigma = \{\emptyset\}$ the above axiom can be re-written as

$$\begin{array}{ll} (K_1) & \forall y\,(y \notin \emptyset) \\ (K_2) & \forall y\,(y \doteq \emptyset)\,. \end{array}$$

Conversely, the theory which has $\langle\{k\}, I_{ii}\rangle$ as unique model (modulo isomorphism) is the following:

$$(K') \qquad \exists x \left(x \doteq x \wedge \forall y\,(y \in x) \wedge \forall y\,(y \doteq x)\right).$$

When $\Sigma = \{\Omega\}$ the above axiom can be re-written as

$$\begin{array}{ll} (K'_1) & \forall y\,(y \in \Omega) \\ (K'_2) & \forall y\,(y \doteq \Omega)\,. \end{array}$$

Both the theories K_1K_2 and $K'_1K'_2$ are decidable. The elimination of quantifier theorem is a consequence of the simplicity of the theory:

Theorem A.2 *Let $\mathcal{L} = \langle\Pi, \{\emptyset\}\rangle$ ($\mathcal{L}' = \langle\Pi, \{\Omega\}\rangle$); then for any formula $\exists x\,(\ell_0 \wedge \cdots \wedge \ell_n)$, where ℓ_i's are literals, exists an open formula ψ such that*

$$K_1K_2(K'_1K'_2) \;\vdash\; \exists x\,(\ell_0 \wedge \cdots \wedge \ell_n) \leftrightarrow \psi\,.$$

Proof. Thanks to axiom K_2 (K'_2), $\exists x\,(\ell_0 \wedge \cdots \wedge \ell_n)$ is equivalent to $(\ell_0 \wedge \cdots \wedge \ell_n)[x/\emptyset]$ $((\ell_0 \wedge \cdots \wedge \ell_n)[x/\Omega])$.

A.2 □

[2] $\bar{k}$ stands for the *king*—see [1].

Deciding the satisfiability of an open conjunction of literals $\ell_0 \wedge \cdots \wedge \ell_n$ is straightforward: first map all the variables to $\emptyset$ (Ω).

Then, in the case of K_1K_2, if a conjunct of the form $\emptyset \neq \emptyset$ or of the form $\emptyset \in \emptyset$ occurs in the disjunction then return unsatisfiable, else return satisfiable.

In the case of $K_1'K_2'$, if a conjunct of the form $\emptyset \neq \emptyset$ or of the form $\emptyset \notin \emptyset$ occurs in the disjunction then return unsatisfiable, else return satisfiable.

This is the proof for

Theorem A.3 *K_1K_2 and $K_1'K_2'$ are both decidable.* □

A.2 The foundation axiom

In this section we will illustrate relations among various axioms stating the well-foundedness of sets. In particular, we will deal with the *acyclicity* axiom schema:[3] for any $n \in \omega$

$$(A) \qquad \neg\exists x_0 \cdots \exists x_n \left(\bigwedge_{i=0}^{n-1} x_i \in x_{i+1} \wedge x_n \in x_0 \right),$$

the *regularity* axiom:

$$(R) \qquad \forall x \exists z \forall y \big(y \in x \to (z \in x \wedge y \notin z) \big),$$

and the syntactical axiom schema (F_3^s), introduced in § 3.2.4: if x is a proper subterm of $t[x]$, then

$$(F_3^s) \qquad \begin{array}{l} x \neq t[x] \\ \text{unless } t \equiv \{s_1, \ldots, s_n \,|\, x\} \text{ and} \\ x \text{ is not a subterm of } s_1, \ldots, s_n. \end{array}$$

Remark A.4 *A few remarks about regularity axiom (R) are due. In classical references (see e.g. [2, 64]) it is stated as follows:*

$$(FA) \qquad \forall x \Big(\forall y\, (y \notin x) \vee (\exists y \in x)(\forall z \in y) z \notin x \Big).$$

Developing restricted quantifiers, it becomes

[3]As usual, the empty conjunction $\bigwedge_{i=0}^{-1} \varphi$ is considered a tautology.

$$(FA) \qquad \forall x \left(\begin{array}{l} \forall y\,(y \notin x)\vee \\ \exists y \forall z \Big((y \in x \wedge z \notin x) \vee (y \in x \wedge z \notin y) \Big) \end{array} \right),$$

while rewriting the implication in (R) as a disjunction, the latter axiom is rewritten as

$$(R) \qquad \forall x\, \exists y\, \forall z \left(z \notin x \vee (y \in x \wedge z \notin y) \right).$$

The equivalence between (R) and (FA) ensues trivially from the following semantical considerations:

- *if x is empty, then (FA) holds trivially thanks to the first disjunct. (R) holds since $z \notin x$ is always true;*

- *let x be non-empty. An element y such that (FA) is satisfied, satisfies also (R). For the other direction, let y be the element satisfying (R). Since x is non-empty, the disjunct $y \in x \wedge z \notin y$ must hold. Hence, since $y \in x$, it satisfies (FA).*

Remark A.5 *We recall that a relation π on a set A is* WELL-FOUNDED *if and only if the following axiom holds*

$$(WF) \qquad (\forall x \subseteq A)\Big(\forall y (y \notin x) \vee (\exists y \in x)(\forall z \in x) \neg \pi z y\Big)$$

Replacing π with $\in$ it is possible to see the similarity with above axiom (FA) (or (R)).

If $\neg(FA)$ then an infinite chain $A \ni y_0 \ni y_1 \ni \cdots$ of elements of A, exists.
Conversely, if (FA) holds, then we are not sure that such an infinite chain does not exist (i.e., when $A = \{\emptyset, y_0, y_1, \cdots\}$: $\emptyset$ is the $\in$-minimal element). However, from the assumption that the property hold for all x (hence for all the subset of A), we can infer the well-foundedness of the membership relation for all the sets considered.

The following technical lemmata will generate a lattice of relations regarding the axioms just presented.

Lemma A.6 *In any model of (W), then (R) implies (A).*

Proof. Let $M = \langle D, I \rangle$ be a model of (W) and (A), and let $a_0, \ldots, a_n \in D$ be such that $a_0 \in^I a_1 \in^I \cdots \in^I a_n \in^I a_0$.

$v = \{a_0, \dots, a_n\}$ belongs to D since M is a model of (W), but there is no z in v such that no y in v belongs to it.

A.6 □

Lemma A.7 *In any model of* (W) *such that the objects of the domain contains only finitely many elements, then* (A) *implies* (R).

Proof. Let $M = \langle D, I \rangle$ be a model of (W) and $\neg(R)$ whose elements are all finite.
Since M is a model of $\neg(R)$, exists an x in D such that, for any $z \in x$, there exists an element y of x such that $y \in^I z$.
Since all elements of D are finite, then such x will have the form $\{v_0, \dots, v_m\}$.
This means that exists an $n \leq m$, and a function $j : \{0, \dots, n\} \rightarrow \{0, \dots, m\}$ such that $\bigwedge_{i=0}^{n-1} v_{j_i} \in^I v_{j_{i+1}} \wedge v_{j_n} \in^I v_{j_0}$.

A.7 □

As corollary from Lemmata A.6 and A.7, we can obtain that, in any model of (W) in which all elements are finite, then (A) and (R) are equivalent.

This is not true in general, as it ensues from:

Lemma A.8 *There are models of* (W) *in which* (A) *does not imply* (R).

Proof. Let $M = \langle D, I \rangle$ be a model of (A) such that a and $\omega(a)$ belongs to D, and $\omega(a) \in^I a$, where $\omega(a)$ is meta-mathematically defined as follows:

$$\begin{cases} u_0 = a \\ u_{i+1} = u_i \cup \{u_i\} \end{cases} \qquad\qquad \omega(a) = \bigcup_{i \in \omega} u_i$$

Observe that there is a membership cycle, but, since it has an infinite length, it does not affect the validity of axiom (A). Conversely, (R) is not satisfied, since no element of $\omega(a)$ has empty intersection with $\omega(a)$ itself.

A.8 □

To deal with axiom F_3^s, we first define the $sub(x, y, n)$, i.e. *x is a subterm of y at depth n*, as follows:

$$\begin{cases} \qquad\qquad sub(x,y,0) \iff y = \{y_1,\dots,y_m \,|\, x\}, m \geq 0 \\ sub(x,\{y_0,\dots,y_m\},n+1) \iff \\ \qquad \exists i \in \{0,\dots,m\} \text{ suchthat } sub(x,y_i,n)\ . \end{cases}$$

(F_3^s) can be re-stated as follows: if $sub(x,y,n)$ holds for some $n > 0$, then $x \neq y$.

Lemma A.9 *(A) implies (F_3^s).*

Proof. We first observe that if x is a non-empty-set and $sub(x,y,n)$ holds for some $n > 0$, then exist $y_1,\dots,y_m$, for some integer m, and $z_0,\dots,z_n$ such that

$$\{y_1,\dots,y_m \,|\, x\} = z_0 \in z_1 \in \cdots \in z_{n-1} \in z_n = y\ .$$

If x and y were equal, we would have

$$z_{n-1} \in \{y_1,\dots,y_m \,|\, x\} = z_0 \in z_1 \in \cdots \in z_{n-1} \in z_n = x\ ,$$

which contradicts (A).

A.9 □

Lemma A.10 *In any model of (W) such that the objects of the domain contains only finitely many elements, then (F_3^s) implies (A).*

Proof. Let $M = \langle D, I\rangle$ be a model of (W) whose elements are all finite.
Assume $x_0 \in^I x_1 \in^I \cdots \in^I x_n \in^I x_0$. Since $x \in y \leftrightarrow \exists z\,(y \doteq \{x \,|\, z\})$ is a theorem for any finite y (see § 3.1.2), we will have

$$\exists z_0 \cdots z_n \left(\begin{array}{l} x_1 = \{x_0 \,|\, z_0\} \wedge x_2 = \{x_1 \,|\, z_1\} \wedge \dots \wedge \\ x_n = \{x_{n-1} \,|\, z_{n-1}\} \wedge x_0 = \{x_n \,|\, z_n\} \end{array} \right) .$$

Hence, for instance,

$$\exists z_0 \cdots z_n\,(x_0 = \{\{\cdots\{\{x_0 \,|\, z_0\} \,|\, z_1\}\cdots\} \,|\, z_n\})$$

which contradicts (F_3^s).

A.10 □

As final corollary of this section, we can state that in all models of (W) such that the objects of the domain contains only finitely many elements, (in particular in $H_\Sigma/\equiv_s$), (A), (R), and (F_3^s) are equivalent.

A.3 Finiteness

In all the thesis the word *finite* has been used with an intuitive meaning. In this section we briefly touch the problem of formally defined such entity.

In 'classical' Zermelo-Fraenkel set theory (see e.g. [64]) there is an axiom ensuring the existence of an infinite set:

$$(\mathit{Inf}) \qquad \exists x \left(\emptyset \in x \wedge (\forall y \in x)(\{y \mid y\} \in x)\right)$$

(the logically simplest form of the infinity axiom can be found in [90, 91]). However, its negation

$$(\neg\mathit{Inf}) \qquad \forall x \left(\emptyset \in x \to (\exists y \in x)(\{y \mid y\} \notin x)\right)$$

is not equivalent to saying that all sets are *finite*.[4] For instance, consider the model $M = \langle D, I\rangle$ of NWE_1E_2 in which $D \supseteq \{[\,t\,] : t \in \tau(\Sigma)\}$ and a representative for the infinite set $\mathit{even} \equiv \{\underline{0}, \underline{2}, \underline{4}, \underline{6}, \ldots\}$ belongs to D. Any element of *even* is a candidate for being the 'y' able to validate $(\neg I)$. Since M is a model of (W) and $\underline{1}, \underline{3}, \underline{5}, \ldots$ all belong to D, any subset of ω which extends *even* with a *finite* number of odd numerals must belong to D itself. However, it is consistent to assume that ω does not belong to D.

In other words, the negation of (I) only states the non-existence of the set ω. If (I) gets modified so as to ensure the existence of some other infinite set, a similar example can easily be found.

There are a wide number of proposals in literature for stating the finiteness of any set (for a deep investigation see [76]). We analyze the one proposed by Tarski in [105]. First the following notions are introduced:

$$\begin{array}{lll} \mathit{irreducible}(a,k) & \text{if and only if} & a \in k \wedge (\forall b \in k)(b \subseteq a \to b \doteq a)\,; \\ \mathit{saturated}(a,k) & \text{if and only if} & a \in k \wedge (\forall b \in k)(a \subseteq b \to a \doteq b)\,. \end{array}$$

$\mathit{irreducible}(a,k)$ –read: a is an *irreducible* element of k– states that no subset of a belongs to k.
$\mathit{saturated}(a,k)$ –read: a is a *saturated* element of k– states that a is a subset of no element of k.

For instance

[4]We recall here *numerals* are defined as $\underline{0} = \emptyset, \underline{n+1} = \{\underline{n} \mid \underline{n}\}$.

- for any set x, x is either *irreducible* or *saturated* in $\{x\}$; x is *saturated* in $\mathcal{P}(x)$;

- $\underline{0}$ and $\underline{n}$ are the *irreducible* and the *saturated* elements of $\{\underline{0}, \underline{1}, \ldots, \underline{n}\}$, respectively;

- there is no *a saturated* in $\omega = \{\underline{0}, \underline{1}, \underline{2}, \ldots\}$; $\underline{0}$ is the only *irreducible* element;

- any $\{i\}$, for $i = 1, \ldots, n$, is either *irreducible* or *saturated* in $\{\{\underline{1}\}, \ldots, \{\underline{n}\}\}$; similarly, any element of the infinite set $\{\{\underline{1}\}, \{\underline{2}\}, \{\underline{3}\}, \ldots\}$ is either *irreducible* or *saturated.*

Definition A.11 *A set x is* FINITE *if any non-empty collection k of its subsets admits at least one irreducible element. Briefly,*

$$(Fin) \quad \forall k \Big(\big((\forall e \in k)(e \subseteq x) \wedge (\exists e \in k)\big) \to \exists a\, (irreducible(a, k))\Big).$$

Assume x be a set with an (intuitively) infinite number of elements. Pick $e_1, e_2, e_3, \cdots \in k$. The set $k = \{x, x \setminus \{e_1\}, x \setminus \{e_1, e_2\}, x \setminus \{e_1, e_2, e_3\}, \ldots\}$ has no *irreducible* elements.
On the other hand, the set $k' = \{\emptyset, \{e_1\}, \{e_1, e_2\}, \{e_1, e_2, e_3\}, \ldots\}$ has no *saturated* elements.

In [105] it is shown

Theorem A.12 *A set x is finite (in the sense of definition A.11) if and only if any non-empty collection of its subsets has at least a saturated element.*

In the proof of Theorem A.12, separation axiom and $\setminus$ operator are used. If the set theory is rich enough to provide an axiomatization for such constructs, then definition A.11 can equivalently be given using the predicate *saturated.* This has been done, for instance, in [106].

A.4 RUQs elimination

Restricted Universal Quantifiers (RUQs) are formulae of the form

$(\forall X \in s)\, \varphi,$

with F an arbitrary formula. This form stands for the quantified implication

$\forall X((X \in s) \to \varphi).$

The usefulness of providing RUQs as part of the representation language has been demonstrated by several authors (e.g. [24, 66]). In fact, RUQs allow basic set-theoretic operations (such as subset, union, intersection and so on) to be expressed in a clear and concise way. In what follows we will show how the language presented so far can be extended in order to encompass RUQs. To wit, RUQs will be introduced in {log} only at the syntactic level, as a convenient notation, without any extension at the semantic level.

Definition A.13 *An* EXTENDED $CLP(\mathcal{S})$ CLAUSE *is a formula:*

$$p(t_1, \ldots, t_n) \leftarrow c \,\Box\, B_1 \,, \cdots ,\, B_m$$

where each B_i can be either

- *an atom, or*
- *a RUQ formula of the form $(\forall X_1 \in s_1) \cdots (\forall X_k \in s_k)G$, G conjunction of atoms, satisfying the following properties:*
  - *the variables $X_1, \ldots, X_k$ can occur only in G (in particular, they are not allowed to occur in $s_1, \ldots, s_k$);*
  - *if $i \neq j$ then $X_i \neq X_j$.*

The two last restrictions ensure that a B_i of the form

$$(\forall X_1 \in t_1) \cdots (\forall X_k \in t_k)G$$

is logically equivalent to $\forall X_1 \cdots \forall X_k(X_1 \in t_1, \cdots, X_k \in t_k \to G)$. (Note that the first restriction, motivated (cf. [24]) by the set finiteness requirement, is implicitly present in [66] since nesting of sets is not allowed there.)

To ease notation, we will not explicitly consider here the separator $\Box$, viewing it as a ',' operator. Moreover, for the same reason, we will deal with sets based on '$\emptyset$'; translation of RUQs with different colors is presented in [36].

For example, by using RUQs, it is easy to define the following set-theoretic operations:

$$\begin{array}{ll} \text{(a)} & \mathsf{subset}(\mathsf{S1},\mathsf{S2}) \leftarrow \\ & \qquad (\forall \mathsf{X} \in \mathsf{S1})(\mathsf{X} \in \mathsf{S2})\,. \\ \text{(b)} & \mathsf{disj}(\mathsf{S1},\mathsf{S2}) \leftarrow \\ & \qquad (\forall \mathsf{X} \in \mathsf{S1})(\forall \mathsf{Y} \in \mathsf{S2})(\mathsf{X} \neq \mathsf{Y})\,. \end{array}$$

where predicate subset tests whether S1 is a subset of S2 and predicate disj tests whether S1 and S2 are disjoint sets.

One might proceed as in [66, 20], by enhancing resolution to deal directly with RUQs. However, both for conceptual simplicity and for soundness concerns we prefer to transform extended Horn clauses into equivalent {log} clauses without RUQs:

RUQs Elimination Algorithm

Let $C = H \leftarrow B_1 \,, \cdots, \, B_k \,, \, B_{k+1} \,, \cdots, \, B_n$ be an extended $CLP(\mathcal{S})$) clause, where $B_1, \ldots, B_k$ ($k \leq n$) are Π_C literals or Π_B atoms, and $B_{k+1}, \ldots, B_n$ are formulae containing RUQs.

1. Replace C by the set of clauses

$$\begin{array}{rcl} I & = & \{\, H \leftarrow c \,\Box\, B_1 \,, \cdots, \, B_k \,, \, D_1 \,, \cdots, \, D_{n-k} \\ & & D_1 \leftarrow B_{k+1} \\ & & \ldots, \\ & & D_{n-k} \leftarrow B_n \,\} \end{array}$$

   where each D_j is an atom $q_j^{new}(X_1^j, \ldots, X_{k_j}^j)$ with q_j^{new} brand new predicate symbol and $\{X_1^j, \ldots, X_{k_j}^j\}$ are all the variables in B_{k+j} which are not quantified by any RUQ of B_{k+j}.

2. Replace each element in I of the form

$$p(t_1, \ldots, t_n) \leftarrow (\forall X_1 \in s_1)(\forall X_2 \in s_2)G$$

by the two clauses:

$$
\begin{array}{l}
p(t_1,\ldots,t_n) \leftarrow (\forall X_1 \in s_1) r(Y_1,\ldots,Y_k) \\
r(Y_1,\ldots,Y_k) \leftarrow (\forall X_2 \in s_2) G
\end{array}
$$

where $Y_1,\ldots,Y_k$ are all the variables (different from X_2) occurring in s_2 or free in G, and r is a new predicate symbol. This step is repeated as long as there are clauses with nested quantifiers in I.

3. Replace each extended Horn clause of the form

$$p(t_1,\ldots,t_n) \leftarrow (\forall X \in \{t'_1,\ldots,t'_m|h\}) G$$

by

$$p(t_1,\ldots,t_n) \leftarrow G[X/t'_1],\cdots,G[X/t'_m]$$

if h is a member-less term, or by

$$
\begin{array}{l}
p(t_1,\ldots,t_n) \leftarrow G[X/t'_1]\,,\cdots,\;G[X/t'_m]\,,\;D \\
D \leftarrow (\forall X \in h) G
\end{array}
$$

if h is a variable, where D is built as described in step 1.

4. Replace each simple extended Horn clause

$$p(t_1,\ldots,t_n) \leftarrow (\forall X \in Y) G[X,Z_1,\ldots,Z_m]\,,$$

where Y is a variable and $X,Z_1,\ldots,Z_m$ ($m \geq 0$) are all the variables occurring in G, by the following three {log} clauses:

$$
\begin{array}{l}
p(t_1,\ldots,t_n) \leftarrow r(Y,Z_1,\ldots,Z_m) \\
r(\emptyset,Z_1,\ldots,Z_m) \leftarrow \\
r(\{A|R\},Z_1,\ldots,Z_m) \leftarrow (A \notin R), G[X/A], r(R,Z_1,\ldots,Z_m)
\end{array}
$$

where r is a new predicate symbol.

For example, the extended Horn clause (a) for the subset operation given at the beginning of this section will be transformed into the equivalent three $CLP(\mathcal{S})$ clauses:

```
subset(S1, S2) ← r(S1, S2).
r(∅, S2) ←.
r({A | R}, S2) ←(A ∉ R), (A ∈ S2), r(R, S2).
```

A.5 Translation of hybrid unification into pure set unification

$\cdots$ *Indeed, one possible view is that integers* <u>*are*</u> *atoms and should not be viewed as sets. Even in this case, one might still wish to prevent the existence of unrestricted atoms. In any case, for the 'genuine' sets, Extensionality holds and the other sets are merely harmless curiosities.* $\cdots$ P. J. Cohen, from [29]

This section establishes a correspondence between systems of equations over nested sets that involve free Herbrand functors (constants, in particular, playing the role of atoms) of the form described and solved in § 4.3.3, and systems over 'genuine' sets. Clearly, solutions are preserved in the translation.

'Genuine' sets—named *pure* sets by us—are those whose construction is ultimately based on the empty set. Such sets have in fact superseded sets with atoms in the early history of set theory, when the Zermelo axiomatization was first subjected to critique (cf. [109]).

We start with a reduction technique by which the *classic* unification problem can be translated into a purely set-theoretic one. Finite, as well as infinite rational terms, can be handled by the proposed technique; in either case, by decoding the solution scheme found after the translation, one can retrieve the most general answer to the original problem.

Careful inspection of the proofs of the Theorems in this section reveals that unification can be performed in polynomial deterministic time in the special case when the sets to be unified are images of standard terms, even though the full unification problem for sets is NP-complete (see § 4.1).

The very same technique can be exploited when the source language itself comprises a set constructor.[5]

We will characterize below a family of 'translations' $tr : \tau(\Sigma \cup \mathcal{V}) \to \mathbf{U}(\mathcal{C} \cup \mathcal{V})$ enforcing the properties of $\{\cdot \mid \cdot\}$, in the following sense: the

[5]The transfer principle that emerges from the proposed translation, carries over to the abstract realm of infinite, even irrational, set terms. In this case, however, the target domain is to comprise sets of infinite rank. Moreover, infinite substitutions would forcibly enter into play.

identities $tr(\{x, y \,|\, z\}) \doteq tr(\{y, x \,|\, z\})$ and $tr(\{x, x \,|\, z\}) \doteq tr(\{x \,|\, z\})$ will hold for all $x, y, z \in \tau(\Sigma \cup \mathcal{V})$. As a consequence, $x =_s y$ (cf. § 2.2) will always imply $tr(x) = tr(y)$. We will manage to have that, conversely, $tr(x) = tr(y)$ imply $x =_s y$; thus, tr will canonically induce a one-to-one function from $\tau(\Sigma \cup \mathcal{V})/\!\!=_s$ into $\mathbf{U}(\mathcal{C} \cup \mathcal{V})$.

Since every x will involve the same variables that occur in $tr(x)$ (in particular, x will be left fixed when it is a variable), the inclusion $tr[\tau(\Sigma)] \subseteq \mathbf{U}(\mathcal{C})$ will hold.

In view of the computability of tr (as well as of $=_s$ and of tr^{-1}), all of this can be summarized by saying that tr constitutes a *set-theoretic semantics* for the terms over Σ. A most important by-product of our definition of tr, however, will go beyond that: it will turn out that any instance of the *unification problem* concerning $\tau(\Sigma \cup \mathcal{V})$ can be solved with reference to $\mathbf{U}(\mathcal{C} \cup \mathcal{V})$. Given arbitrary t_1, t_2 from $\tau(\Sigma \cup \mathcal{V})$, one can determine an exhaustive set $\{\mu_1, \ldots, \mu_n\}$ of mgus of $tr(t_1) \doteq tr(t_2)$. If this set is empty, t_1, t_2 are not unifiable; else $\{tr^{-1}(\mu_1), \ldots, tr^{-1}(\mu_n)\}$ is a complete set of unifiers for $t_1 \doteq t_2$, where $tr^{-1}(\mu)$ denotes the substitution sending each variable X into the counter-image $tr^{-1}(X\mu)$.

For any $f \in \Sigma \setminus \{\{\cdot \,|\, \cdot\}\}$ let c_f be a distinctive non-negative integer. A family of monomorphisms from $\tau(\Sigma \cup \mathcal{V})/\!\!=_s$ to $\mathbf{U}(\mathcal{C} \cup \mathcal{V})$, indexed by a non-negative integer k, can be defined as follows:

$$\left\{\begin{array}{rcl} tr(X) & = & X\,, \\ tr(f(t_1, \ldots, t_n)) & = & \{\underline{k + c_f}, \{\underline{1}, \underbrace{\{\cdots\{}_{k}\, tr(t_1)\, \underbrace{\}\cdots\}}_{k}\}, \ldots, \\ & & \{\underline{n}, \underbrace{\{\cdots\{}_{k}\, tr(t_n)\, \underbrace{\}\cdots\}}_{k}\}\}\,, \\ tr(\{t_1 \,|\, t_2\}) & = & tr(t_2) \cup \{\, \underbrace{\{\cdots\{}_{k}\, tr(t_1)\, \underbrace{\}\cdots\}}_{k}\,\}\,, \end{array}\right.$$

In the rest of the section, we will prove that tr satisfies the required isomorphism property for mgus, be the starting universe well founded or not.

First we want to point out the meaning of the parameter k. If the set of generators $\mathcal{C}$ is the set $\{\underline{0}, \underline{1}, \underline{2}, \ldots\}$ then k can be left free to take any non-negative value.

If, on the contrary, $\mathcal{C}$ is empty, then we take as $\underline{0}, \underline{1}, \underline{2}, \ldots$ the *numerals* defined *à la* von Neumann: $\underline{0} = \emptyset$, $\underline{n+1} = \underline{n} \cup \{\underline{n}\}$; moreover, in this case we require k to be at least 2 in order that tr be one-to-one. In particular we prove all the claims in this framework and assuming k to be exactly 2.

We will use t_1, t_2 to denote the terms to be unified, and, in general $t, t_1, t_2, \dots$ as generic $(\Sigma \cup \mathcal{V})$-terms. ℓ_i and r_i will be used to denote $(\Sigma \cup \mathcal{V})$-terms when they occur as left-hand and right-hand side of an equation, respectively. We will use $s, s_1, s_2, \dots$ to denote *sets*, and standard operators on sets will be employed with their usual meaning.

The following technical Lemma, whose proof is straightforward, is the key for proving that tr is one-to-one.

Lemma A.14 *The following facts hold:*
(a) $\underline{1}$ *(i.e.* $\{\emptyset\}$*) is the only* singleton *numeral;*
(b) $\{\{s\}\}$ *is not a numeral, for any set* s*;*
(c) if $n > 0$ *then neither* $\{\underline{n}\}$ *nor* $\{\underline{n}, \{\{s\}\}\}$ *is a numeral;*
(d) if $n \neq 1$ *then* $\{\underline{n}\} \neq \{\{s\}\}$ *for any set* s*.*

As corollary of Lemma A.14, it is easy to see that there is no way to confuse:

- a numeral with a set $\{\underline{n}, \{\{s\}\}\}$, if $n > 0$;
- $\{\underline{n_1}, \{\{s_1\}\}\}$ with $\{\underline{n_2}, \{\{s_2\}\}\}$, provided $n_1, n_2 > 0$, $n_1 \neq n_2$;
- a numeral greater than 1 with a set $\{\{s\}\}$;
- $\{\underline{n}\}$ with $\{\{s\}\}$ for any set s.

This suggests the following unambiguous *decoding procedure* implementing tr^{-1}: let s be a set;

```
 if $s$ is a variable **then** $s$ is its own image
elseif a set $s_2$ such that $\{\{s_2\}\} \in t$ exists **then**
    • $s$ should be of the form $tr(\{t_2 \mid t_1\})$;
    • decode $s_2$ in order to find $t_2$;
    • decode $s \setminus \{\ \{\{s_2\}\}\ \}$ in order to find $t_1$,
elseif a numeral $\underline{2 + c_f} \in s$ exists and it is the only numeral in $s$ **then**
```

- s should be of the form $tr(f(t_1, \ldots, t_n))$, where $n = ar(f)$;
- $s \setminus \{\, \underline{2 + c_f} \,\}$ should be of the form $\{\{\underline{1}, \{\{s_1\}\}\}, \ldots, \{\underline{n}, \{\{s_n\}\}\}\}$;
- decode s_i in order to find t_i, for $i = 1, \ldots, n$,

else the term is not of the form $tr(t)$.

Each 'decode' call stands for 'if it is possible then decode else fail'.

It is easy to verify by structural induction that for any term t and any grounding substitution γ $tr^{-1}(tr(t))\gamma =_s t\gamma$. Assume, moreover, that $tr(t_1) = tr(t_2)$; since tr^{-1} uniquely determines a term, this means that tr is one-to-one.

Since $tr(X) = X$ for all variables, it is obvious to see that, in any of the above-mentioned cases, if μ is an mgu of t_1 and t_2, then $tr(\mu)$ is an mgu of $tr(t_1)$ and $tr(t_2)$. The converse is discussed below.

The standard (well-founded) case

We will prove below the desired isomorphism for mgus induced by the function tr, when the domain of tr is exactly the universe $H_\Sigma / \equiv_s$ (cf. § 3.1).

Theorem A.15 *For any $t_1, t_2 \in \tau(\Sigma \cup \mathcal{V})$ and for any mgu μ of $tr(t_1) \doteq tr(t_2)$ in a well-founded set theory with extensionality, $tr^{-1}(\mu)$ is an mgu of $t_1 \doteq t_2$.*

Proof. We prove a stronger claim:
If $\mathcal{E}$ is a system of equations $tr(\ell_1) \doteq tr(r_1) \wedge \ldots \wedge tr(\ell_k) \doteq tr(r_k)$ and μ is an mgu of $\mathcal{E}$, then $tr^{-1}(\mu)$ is an mgu of $\ell_1 \doteq r_1 \wedge \ldots \wedge \ell_k \doteq r_k$.

Let p be the number of occurrences of functional symbols $\emptyset$ and $\{\cdot \,|\, \cdot\}$ needed to write $tr(\ell_1), tr(r_1), \ldots, tr(\ell_k), tr(r_k)$ (here we make a 'meta' use of $\emptyset$ and $\{\cdot \,|\, \cdot\}$). If $\mathcal{E}$ is satisfiable, it is easy to see (cf. 4.2.4) that there exists a function $lev : FV(\mathcal{E}) \to \omega$, extended to terms as follows

$$\begin{array}{rcl} lev(\emptyset) & = & 0 \\ lev(\{t_1 \,|\, t_2\}) & = & \max\{lev(t_2), 1 + lev(t_1)\} \end{array}$$

and fulfilling the condition

$$(*) \qquad 0 \leq lev(tr(\ell)) \doteq lev(tr(r)) \leq p \quad \text{for any } tr(\ell) \doteq tr(r) \text{ in } \mathcal{E}.$$

We prove the claim by induction on the parameter $\hat{s}(\mathcal{E})$, defined to be the (finite) $(p+1)$-tuple of non-negative integers

$$[\mid \{e\, in\, \mathcal{E} : lev(e) = p\} \mid, \cdots, \mid \{e\, in\, \mathcal{E} : lev(e) = 0\} \mid]$$

ordered lexicographically.

For any equation $tr(\ell) \doteq tr(r)$ in $\mathcal{E}$, $tr(\ell)$ and $tr(r)$ can be in one of the forms

1. X;
2. $\{\underline{c_f + 2}, \{\underline{1}, \{\{s_1\}\}\}, \ldots, \{\underline{n}, \{\{s_n\}\}\}\}$, $n = ar(f)$;
3. $\{\{\{s_1\}\}, \ldots, \{\{s_h\}\} \mid R\}$ (i.e. $\{\{\{s_1\}\}, \ldots, \{\{s_h\}\}\} \cup R$);
4. $\{\{\{s_1\}\}, \ldots, \{\{s_h\}\}, \underline{c_f + 2}, \{\underline{1}, \{\{s'_1\}\}\}, \ldots, \{\underline{n}, \{\{s'_n\}\}\}\}$, $k > 0$, $n = ar(f)$.

Let $\mathcal{E}$ be $tr(\ell_1) \doteq tr(r_1) \wedge \ldots \wedge tr(\ell_k) \doteq tr(r_k)$. Suppose, without loss of generality, that $tr(\ell_1) \doteq tr(r_1)$ is selected.
Base) $\langle 1, 1\rangle$. $\mathcal{E}$ has the form $X \doteq Y$. Trivially $tr^{-1}(\mu) = \mu = [X/Y]$.
Step)

Cases $\langle 1, 2\rangle, \langle 1, 3\rangle, \langle 1, 4\rangle$. $tr(\ell)$ is the variable X.

Suppose X does not occur in $tr(r)$. Then $\hat{s}(tr(\ell_2) \doteq tr(r_2) \wedge \ldots \wedge tr(\ell_k) \doteq tr(r_k)[X/tr(r)])$ is strictly less than $\hat{s}(\mathcal{E})$. Note that $tr(t)[X/tr(r)] = tr(t[X/r])$, hence for any mgu σ of $4tr(\ell_2) \doteq tr(r_2) \wedge \ldots \wedge tr(\ell_k) \doteq tr(r_k))[X/tr(r)]$, by induction hypothesis, $tr^{-1}(\sigma)$ is an mgu of $(\ell_2 \doteq r_2 \wedge \ldots \wedge \ell_k \doteq r_k)[X/r]$.

Since any mgu μ of $\mathcal{E}$ has the form $[X/tr(r)\sigma] \cup \sigma$ then $tr^{-1}(\mu) = [X/r tr^{-1}(\sigma)] \cup tr^{-1}(\sigma)$ is an mgu of $\ell_1 \doteq r_1 \wedge \ldots \wedge \ell_n \doteq r_n$.

If X occurs in $tr(r)$ then the latter should have the form (3) with $R \equiv X$ and X does not occur elsewhere in $tr(r)$. By renaming the occurrence of X in $tr(r)$ with a new variable N we can repeat the above reasoning. Note that N is a subset of X, hence we can extend lev to N fulfilling $lev(N) \leq lev(X)$.

If X occurs elsewhere in $tr(r)$, then $\mathcal{E}$ is not satisfiable; hence the claim trivially holds.

Case $\langle 2,2\rangle$**.** $tr(\ell) \equiv \{\underline{c_f+2}, \{\underline{1}, \{\{s_1\}\}\}, \ldots, \{\underline{n}, \{\{s_n\}\}\}\}$, $n = ar(f)$, and
$tr(r) \equiv \{\underline{c_{f'}+2}, \{\underline{1}, \{\{s'_1\}\}\}, \ldots, \{\underline{n'}, \{\{s'_{n'}\}\}\}\}$, $n' = ar(f')$.

If f is different from f' or n is different from n', then Lemma A.14 and extensionality ensure that $tr(\ell)$ and $tr(r)$ cannot be unified.

If $f \equiv f'$ and $n \equiv n'$, by Lemma A.14, γ is a unifier of $tr(\ell) \doteq tr(r)$ if and only if it is a unifier of $s_1 \doteq s'_1 \wedge \ldots \wedge s_n \doteq s'_n$. Consider $\mathcal{E}' = s_1 \doteq s'_1 \wedge dots \wedge s_n \doteq s'_n \wedge tr(\ell_2) \doteq tr(r_2) \wedge \ldots \wedge tr(\ell_k) \doteq tr(r_k)$; it may be the case that *lev* does not fulfill condition $(*)$ for $\mathcal{E}'$; however it is easy to see that, if $\mathcal{E}$ is satisfiable, an initial level function satisfying $(*)$ also for $\mathcal{E}$ does exists. Since $\hat{s}(\mathcal{E}')$ is fewer than $\hat{s}(\mathcal{E})$ we can apply induction hypothesis.

Case $\langle 3,3\rangle$**.** $tr(\ell) \equiv \{\{\{s_1\}\}, \ldots, \{\{s_h\}\} \,|\, R\}$,
$tr(r) \equiv \{\{\{s'_1\}\}, \ldots, \{\{s'_{h'}\}\} \,|\, R'\}$.

By Lemma A.14, γ is a unifier of $tr(\ell) \doteq tr(r)$ if and only if it is a unifier of

- $s_{i_1} \doteq s'_{j_1} \wedge \ldots \wedge s_{i_m} \doteq s'_{j_m} \wedge$
  $R \doteq \{\{\{s'_{h'_1}\}\}, \ldots, \{\{s'_{h'_l}\}\} \,|\, N\} \wedge$
  $R' \doteq \{\{\{s_{h_1}\}\}, \ldots, \{\{s_{h_p}\}\} \,|\, N\}$,
  with $m, l, p \geq 0$, if R is different from R';
- $s_{i_1} \doteq s'_{j_1} \wedge \ldots \wedge s_{i_m} \doteq s'_{j_m} \wedge R \doteq \{\{\{s'_{h'_1}\}\}, \ldots, \{\{s'_{h'_l}\}\},$
  $\{\{s_{h_1}\}\}, \ldots, \{\{s_{h_p}\}\} \,|\, N\}$
  with $m, l, p \geq 0$, if $R \equiv R'$.

If we replace $tr(\ell) \doteq tr(r)$ in $\mathcal{E}$ with $s_{i_1} \doteq s'_{j_1} \wedge \ldots \wedge s_{i_m} \doteq s'_{j_m}$ and apply the substitution
$[R/\{\{\{s'_{h'_1}\}\}, \ldots, \{\{s'_{h'_l}\}\} \,|\, N\}, R'/\{\{\{s_{h_1}\}\}, \ldots, \{\{s_{h_p}\}\} \,|\, N\}]$, or
$[R/\{\{\{s'_{h'_1}\}\}, \ldots, \{\{s'_{h'_l}\}\}, \{\{s_{h_1}\}\}, \ldots, \{\{s_{h_p}\}\} \,|\, N\}]$,
respectively, eventually adapting the level function as shown in the previous case and extending it for N fulfilling

$$lev(N) \leq \min\{lev(R), lev(R')\},$$

we can apply the induction hypothesis to the obtained system.

Case $\langle 3,4\rangle, \langle 4,4\rangle$**.** Proof are similar (just a bit longer) than the ones of the previous two cases.

$\langle 2,3\rangle, \langle 2,4\rangle$**.** In this case there are no unifiers for $tr(\ell_1) \doteq tr(r_1)$, hence there are no mgus for $\mathcal{E}$.

Remaining cases. For any other case $\langle i,j\rangle$, refer to the corresponding case $\langle j,i\rangle$.

A.15 □

Analyzing the proof of the Theorem, it ensues that for any mgu $\mu = [X_1/s_1, \ldots, X_n/s_n]$ of $tr(t_1) \doteq tr(t_2)$, then $s_1 = tr(r_1), \ldots, s_n = tr(r_n)$ for some terms $r_1, \ldots, r_n$. The corresponding mgu

$$tr^{-1}(\mu) = [X_1/tr^{-1}(s_1), \ldots, X_n/tr^{-1}(s_n)]$$

can be easily obtained using the described procedure for computing tr^{-1}.

The rational (non-well-founded) case

The definition of *tr* makes sense even for infinite Σ-terms. Among them, the class of rational terms (i.e. infinite terms that can be represented by a finite number of equations between $\Sigma \cup \mathcal{V}$-terms).

Assume $\{\cdot \mid \cdot\}$ does not belong to Σ and consider the rational tree theory presented in [74] (there named E_r^*), consisting of the axiom schemata

$$\begin{array}{ll}
(1) & f(x_1, \ldots, x_n) \doteq f(y_1, \ldots, y_n) \to x_1 \doteq y_1 \land \cdots \land x_n \doteq y_n \\
(2) & f(x_1, \ldots, x_m) \not\doteq g(y_1, \ldots, y_n) \\
(3) & \forall \bar{y}\, \exists! x_1, \ldots, x_n\, (x_1 \doteq t_1[\bar{x}, \bar{y}] \land \cdots \land x_n \doteq t_n[\bar{x}, \bar{y}])\,.
\end{array}$$

(that is, axioms schemata (F_1), (F_2), and (F_4) of § 3.2). Axiom schema (3) reinforces (with uniqueness) the corresponding axiom of [31]:

A system of equations of the following form[6] *has at least one tree-solution:* $X_1 \doteq t_1 \land \ldots \land X_n \doteq t_n$*, where the* X_i*'s are distinct variables and the* t_i*'s are any terms.*

[6]Defined as *solved form.*

A few considerations about the meaning of mgu of two rational trees are due.

The mgu of $e \equiv X \doteq f(X)$ is the substitution $\mu = [X/f(f(f(\cdots)))]$. In other words we assign the rational term represented by e to X.

Analogously, the mgu of $tr(e) \equiv X \doteq \{\underline{c_f + 2}, \{\underline{1}, \{\{X\}\}\}\}$ is the substitution

$$\begin{aligned}\mu' \quad = \quad & [X/\{\underline{c_f + 2}, \{\underline{1}, \{\{ \\ & \qquad\qquad \{\underline{c_f + 2}, \{\underline{1}, \{\{ \\ & \qquad\qquad\quad \{\underline{c_f + 2}, \{\underline{1}, \{\{ \cdots \}\}\}\} \ \cdots \} \,].\end{aligned}$$

μ' is the function assigning the non well-founded set represented by e to X (μ' is, in a sense, $tr(\mu)$).

Notice that the uniqueness forced by axiom schema (3) has as counter-part the anti-foundation axiom **AFA**. For instance,

$$\mu = [X/f(f(f(\cdots))), Y/f(f(f(\cdots)))]$$

is an mgu for the system

$$\mathcal{E} = X \doteq f(Y) \wedge Y \doteq f(X).$$

This means (existence and uniqueness) that $X = Y$. Moreover,

$$tr(\mathcal{E}) = X \doteq \{\underline{c_f + 2}, \{\underline{1}, \{\{Y\}\}\}\} \wedge Y = \{\underline{c_f + 2}, \{\underline{1}, \{\{X\}\}\}\}$$

forces X to be equal to Y by **AFA**.

We recall here the solved form algorithm for rational terms described in [31]

1.	$X \doteq X \wedge \mathcal{E}$	$\Rightarrow$	$\mathcal{E}$
2.	$X \doteq Y \wedge \mathcal{E}$	$\Rightarrow$	$\mathcal{E}[X \leftarrow Y]$
3.	$t \doteq X \wedge \mathcal{E}$	$\Rightarrow$	$X \doteq t \wedge \mathcal{E}$
4.	$X \doteq t_1 \wedge X \doteq t_2 \wedge \mathcal{E}$	$\Rightarrow$	$X \doteq t_1 \wedge t_1 \doteq t_2 \wedge \mathcal{E}$
5.	$f(t_1, \ldots, t_m) \doteq g(t'_1, \ldots, t'_n) \wedge \mathcal{E}$	$\Rightarrow$	`fail`
6.	$f(t_1, \ldots, t_n) \doteq f(t'_1, \ldots, t'_n) \wedge \mathcal{E}$	$\Rightarrow$	$t_1 \doteq t'_1 \wedge \cdots \wedge t_n \doteq t'_n \wedge \mathcal{E}$

where X and Y denote variables, t denotes any non variable term, t_i, t'_i denote any term, f and g are different functional symbols.

Let Σ be a signature and $\mathcal{V}$ a set of variables.

Theorem A.16 *Let t_1 and t_2 be $\Sigma \cup \mathcal{V}$-terms. μ is the (unique) mgu of t_1 and t_2 with respect to E_r^* iff $tr(\mu)$ is the (unique) mgu of $tr(t_1)$ and $tr(t_2)$ with respect to any theory having the universe of hypersets ([3]) as a model.*

Proof. We prove that if $\mathcal{E}_0, \ldots, \mathcal{E}_m$ is a (terminating) computation of the unification algorithm above, then $tr(\mathcal{E}_0), \ldots, tr(\mathcal{E}_m)$ is the corresponding deterministic computation performed by a unification algorithm for hypersets. We prove the claim by induction on m.

In the base case ($m = 0$, i.e. $\mathcal{E}_0$ is in solved form) the relations between axiom (3) of [74] and the anti-foundation axiom **AFA** are discussed. A few preliminaries are due.

Let $\mathcal{E} = X_1 \doteq t_1 \wedge \ldots \wedge X_n \doteq t_n$ be a solved form system of equations (i.e. $X_i \not\equiv X_j$ when $i \neq j$); X_i may occur in t_j for any $i, j \in \{1, \ldots, n\}$.

Let $Y_1, \ldots, Y_k$ (abbr. $\bar{Y}$) be the variables in $t_1, \ldots, t_n$ (abbr. $\bar{t}$) distinct from $X_1, \ldots, X_n$ (abbr. $\bar{X}$). With $[\bar{X}/\bar{X}^{(k)}]$ we denote the substitution $[X_1/X_1^{(k)}, \ldots, X_n/X_n^{(k)}]$, where for any i, j, k X_i and $X_j^{(k)}$ are distinct variables, and, moreover, if $\langle i, j\rangle \neq \langle h, k\rangle$, then $X_j^{(i)} \neq X_k^{(h)}$.

Let $\mathcal{E}^\omega$ be the system $\bar{X} \doteq \bar{t}[\bar{X}/\bar{X}^{(1)}] \wedge \bar{X}^{(1)} \doteq \bar{t}[\bar{X}/\bar{X}^{(2)}] \wedge \bar{X}^{(2)} \doteq \bar{t}[\bar{X}/\bar{X}^{(3)}] \wedge \ldots$. Equivalence between $\mathcal{E}$ and $\mathcal{E}^\omega$ holds by standard equality axioms. Let $\theta_{\mathcal{E}^\omega}$ be the substitution

$$[\bar{X}^{(1)}/\bar{t}[\bar{X}/\bar{X}^{(2)}], \bar{X}^{(2)}/\bar{t}[\bar{X}/\bar{X}^{(3)}], \ldots] \,.$$

$\mathcal{E}_0$ is a solved form system. For any tuple of trees $\bar{r}$ assigned to $\bar{Y}$, axiom (3) ensures existence and uniqueness of a (tree) solution γ for $\bar{X}$. Such a solution should be exactly

$$\gamma \;=\; \mathcal{E}_0^\omega \theta_{\mathcal{E}_0^\omega} \theta_{\mathcal{E}_0^\omega} \theta_{\mathcal{E}_0^\omega} \cdots |_{X_1 \cdots X_n}$$

Consider $tr(\mathcal{E}_0)$; as shown in example (1.8) of [3], for any set assignment to $\bar{Y}$, the existence and uniqueness of a (set) solution for $(tr(\mathcal{E}_0))^\omega$, is a simple consequence of **AFA**. It is easy to see that $(tr(\mathcal{E}_0))^\omega$ is the same as $tr(\mathcal{E}_0^\omega)$. As above, the (unique) solution to the last system will be

$$\delta \;=\; (tr(\mathcal{E}_0))^\omega \theta_{(tr(\mathcal{E}_0))^\omega} \theta_{(tr(\mathcal{E}_0))^\omega} \theta_{(tr(\mathcal{E}_0))^\omega} \cdots |_{X_1 \cdots X_n}$$

It is a matter of routine to see that, provided γ is computed mapping $\bar{Y}$ into $\bar{r}$, if δ is computed mapping $\bar{Y}$ into $tr(\bar{r})$, then δ is exactly $tr(\gamma)$.

For the induction step, let $\mathcal{E}_0 = e_1 \wedge \ldots \wedge e_k$; assume that the unification algorithm terminates in $m+1$ step. Without loss of generality, assume moreover that the first equation is the one selected at the first step. The only two non trivial cases are the following:

- $e_1 \equiv f(t_1, \ldots, t_m) \doteq g(t'_1, \ldots, t'_n)$. In this case it follows from the freeness axioms that $\mathcal{E}_0$ is equivalent to `false`. On the other hand, it is a simple consequence of Lemma A.14 that $tr(f(t_1, \ldots, t_m))$ cannot be unified with $tr(g(t'_1, \ldots, t'_n))$.
- $e_1 \equiv f(\ell_1, \ldots, \ell_n) \doteq f(r_1, \ldots, r_n)$. By induction hypothesis $tr(\mathcal{E}_m)$ is the solved form of $tr(\mathcal{E}_1) = tr(\ell_1) \doteq tr(r_1) \wedge \ldots \wedge tr(\ell_n) \doteq tr(r_n) \doteq tr(e_2) \doteq \ldots \doteq tr(e_k)$. We only need to prove that $tr(\mathcal{E}_1)$ is equivalent, from the hyperset point of

  view, to $tr(\mathcal{E}_0)$. This follows from Lemma A.14 which ensures, in particular, that

$$\begin{array}{lcl} \{\underline{c_f+2}, & & \{\underline{c_f+2}, \\ \{\underline{1}, \{\{tr(\ell_1)\}\}\}, & \doteq & \{\underline{1}, \{\{tr(r_1)\}\}\}, \\ \quad\vdots & & \quad\vdots \\ \{\underline{n}, \{\{tr(\ell_n)\}\}\}\} & & \{\underline{n}, \{\{tr(r_n)\}\}\}\} \end{array}$$

  if and only if

$$\begin{array}{rcl} tr(\ell_1) & \doteq & tr(r_1), \\ & \vdots & \\ tr(\ell_n) & \doteq & tr(r_n) \end{array}$$

A.16 □

To complete this part we briefly discuss the case in which t_1 and t_2 are (blended) terms written in an enriched language containing the binary set-theoretic operator $\{\cdot \mid \cdot\}$.

A unification algorithm for such case has been presented in § 4.3.3; as already said, its skeleton is the same as that of [31] and the most peculiar feature is the treatment of equations of the form

$$\{t_0, \ldots, t_m \,|\, t\} \;\doteq\; \{t'_0, \ldots, t'_n \,|\, t'\}$$

This kind of equations can be solved adapting the technique employed to deal with the other cases. In practice we want to reduce

$$\{t_0, \ldots, t_m \,|\, t\} \doteq \{t'_0, \ldots, t'_n \,|\, t'\}$$

to a list of equations involving $t_0, \ldots, t_m, t, t'_0, \ldots, t'_n, t'$ only. To this end turns out to be convenient to introduce a certain amount of non-determinism taking special care of the following cases:

1. t and t' are both of the form $f(\ldots)$, f different from $\{\cdot\,|\,\cdot\}$;

2. one from t and t' is $f(\ldots)$ with f different from $\{\cdot\,|\,\cdot\}$ and the other is a variable;

3. t and t' are both variables.

As we have seen in Chapter 4, the last case is the most challenging one, since a wrong sequence of non-deterministic choices can easily cause non-termination.

Moreover, even without entering any detail of a specific unification algorithm based on the above outlined ideas, we can argue that a result corresponding to Theorem A.16 holds also for the blended case. The main motivation for this fact is the observation that whatever is the choice made to reduce

$$\{t_0, \ldots, t_m \,|\, t\} \;\doteq\; \{t'_0, \ldots, t'_n \,|\, t'\}$$

in $\mathcal{E}$, the corresponding choice to reduce

$$\begin{array}{l} \{\, \{\{tr(t_0)\}\}, \ldots, \{\{tr(t_m)\}\} \,|\, tr(t) \,\} \;\doteq \\ \{\{\{tr(t'_0)\}\}, \ldots, \{\{tr(t'_n)\}\} \,|\, tr(t') \,\}. \end{array}$$

in $tr(\mathcal{E})$ can be performed. The correspondence between the mgus μ for $\mathcal{E}$ and $tr(\mu)$ for $tr(\mathcal{E})$ continues to hold in view of Lemma A.14.

An alternative 3-step reduction

We present below another injective mapping from $\tau(\Sigma \cup \mathcal{V})$ to $\mathbf{U}(\mathcal{V})$. Let Σ be $\{\{\cdot \,|\, \cdot\}, f_1, \dots, f_k\}$. We split such a definition into three steps: this creates two interesting intermediate equivalence results.

Colored Sets of Colored Sets. As first step we define the function

$$tr_1 : \tau(\Sigma \cup \mathcal{V}) \to \tau(\{\{\cdot \,|\, \cdot\}, c_0, \dots, c_A, c_{A+1}, \dots, c_{A+k}\} \cup \mathcal{V})$$

where $A = \max\{ar(f) : f \in \Sigma \setminus \{\{\cdot \,|\, \cdot\}\}\}$.

$$\begin{array}{rcl} tr_1(X) & = & X \\ tr_1(f_i(t_1, \dots, t_n)) & = & \{\{tr_1(t_1)\}_{c_1}, \dots, \{tr_1(t_n)\}_{c_n}\}_{c_{A+i}} \\ tr_1(\{t \,|\, s\}) & = & \{\{tr_1(t)\}_{c_0} \,|\, tr_1(s)\} \end{array}$$

Observe that any element in $tr_1[\tau(\Sigma)]$ is the concrete form of a hereditarily finite set based on a *color*. $c_0, \dots, c_{A+k}$ are the possible (distinct) colors of such sets. Note that such new constants can occur in a set only as colors, not as elements.

Pure Sets with *ur*-elements. The second translation step is to encode the above objects into a another universe of terms representing sets. This time sets are not colored; however non-sets (atoms) can also occur as elements:

$$\begin{array}{rcl} tr_2(X) & = & X \\ tr_2(c_i) & = & \{c_i\} \\ tr_2(\{t \,|\, s\}) & = & \{tr_2(t) \,|\, tr_2(s)\} \end{array}$$

Observe that tr_2 is defined from $\tau(\{\{\cdot \,|\, \cdot\}, c_0, \dots, c_A, c_{A+1}, \dots, c_{A+k}\} \cup \mathcal{V})$ into itself.

Pure Sets. In the third step we perform the elimination of the $A + k$ urelements in a similar way to the one used in early works on set theory. Briefly, we raise the *rank* of any set by $A + k$ and assign to each constant c_i the set $\underbrace{\{\cdots\{}_{i}\emptyset\underbrace{\}\cdots\}}_{i}$. In this way all the constants are forced to remain distinct. Moreover, any other set cannot be identified with such objects since its rank has been sufficiently raised.

$$\begin{aligned}
tr_3(X) &= X\\
tr_3(\emptyset) &= \emptyset\\
tr_3(c_i) &= \underbrace{\{\cdots\{}_{i}\,\emptyset\,\underbrace{\}\cdots\}}_{i}\\
tr_3(\{t \mid s\}) &= \underbrace{\{\cdots\{}_{A+k+1}\{tr_3(t) \mid tr_3(s)\}\underbrace{\}\cdots\}}_{A+k+1}
\end{aligned}$$

The complete coding function tr is simply the composition of tr_1, tr_2, and tr_3.

A.6 Benchmark tables

The following tables report some numerical values for the functions computing the minimal number of m.g.u.'s for the sample problems presented in § 4.4. The number on the 'x' axis denotes the value for m (the first argument). Partially defined matrices are symmetrical.

(1)	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2		2	6	14	30	62	126
3			6	36	150	540	1806
4				24	240	1560	8400
5					120	1800	16800
6						720	15120
7							5040

(2)	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2		2	6	14	30	62	126
3			15	48	165	558	1827
4				184	680	2664	11032
5					2945	13080	59605
6						63756	320292
7							1748803

Problem (3) (numerically equal to problem (4)) would require a three-dimensional matrix to represent its values. Assume $k = 3$:

(3)	1	2	3	4	5	6	7
1	7	19	61	223	877	3559	14581
2		56	195	746	3093	13808	65391
3			705	2859	12681	60231	302829
4				12226	56891	284286	1510483
5					277091	1448325	8044117
6						7888698	45590823
7							273498973

(5)	1	2	3	4	5	6	7
1	2	2	2	2	2	2	2
2	4	12	28	60	124	252	508
3	6	30	126	462	1566	5070	15966
4	8	56	344	1880	9368	43736	195224
5	10	90	730	5370	36250	228090	1359130
6	12	132	1332	12372	106452	856212	6505812
7	14	182	2198	24710	259574	2562182	23928758

(6)	1	2	3	4	5	6	7
1	2	2	2	2	2	2	2
2	5	12	28	60	124	252	508
3	10	42	144	486	1596	5106	16008
4	19	126	584	2584	11208	48248	205864
5	36	360	2200	11930	63000	330450	1733000
6	69	1016	8118	52740	325812	1983084	12073836
7	134	2870	29876	231518	1641444	11310530	77511140

(7)	1	2	3	4	5	6	7
1	2	4	8	16	32	64	128
2		11	30	85	248	735	2194
3			103	356	1269	4678	17735
4				1441	5940	25237	110668
5					27631	131142	640513
6						685507	3660958
7							21169037

(8)	1	2	3	4	5	6	7
1	4	9	18	35	68	133	262
2		39	131	413	1185	3459	10071
3			652	2811	11402	44983	175224
4				15937	82499	409897	1997795
5					524056	3133773	18217350
6						21998671	148144723
7							1136372140

Solutions Computed by a Naive Algorithm (see § 4.4.2)

(2)	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2		5	13	29	61	125	253
3			73	301	1081	3613	11953
4				2069	11581	57749	268381
5					95401	673261	4306681
6						6487445	55213453
7							610093513

(7)	1	2	3	4	5	6	7
1	4	10	22	46	94	190	382
2	13	67	265	931	3073	9787	30505
3	46	424	2692	14356	69436	316324	1386172
4	193	2845	26689	201637	1343353	8259805	48109009
5	976	21046	273946	2785306	24436786	194636506	1449663106
6	5869	173215	2982457	39232711	437961529	4380170455	40526990857
7	41098	1582372	34748680	573495616	3913855304	65037766320	834652259744

(8)	1	2	3	4	5	6	7
1	4	10	22	46	94	190	382
2		52	208	736	2440	7792	24328
3			1372	7516	37012	170668	754132
4				60316	418996	2653036	15780916
5					3964684	33340420	258420172
6						363503932	3587040388
7							44280657292

Bibliography

[1] Abbott, E. A. *Flatland. A Romance of Many Dimensions.* (Italian edition) Adelphi, Milan, 1993.

[2] Abian, A. *The Theory of Sets and Transfinite Arithmetic.* W. B. Saundners Company, Philadelphia and London, 1965.

[3] Aczel., P. *Non-well-founded sets.*, vol. 14 of *Lecture Notes, Center for the Study of Language and Information.* Stanford, 1988.

[4] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[5] Aho, A. V., Sethi, R., and Ullman, J. D. *COMPILERS: Principles, Techniques, and Tools.* Addison-Wesley, 1985.

[6] Aiken, A. Set Constraints: Results, Applications and Future Directions. Technical report, University of California, Berkeley, 1994.

[7] Aliffi, D., Dovier, A., Omodeo, E. G., and Rossi, G. Unification of hyperset terms. Unpublished proc. of Workshop on Logic Programming with Sets, in conjunction with ICLP'93, June 1993.

[8] Apt, K. R. Introduction of Logic Programming. In *Handbook of Theoretical Computer Science, vol. B: Formal Models and Semantics*, J. van Leeuwen, Ed. Elsevier and The MIT Press, 1990.

[9] Apt, K. R., and Bol, R. Logic Programming and Negation: a Survey. *Journal of Logic Programming 19,20* (1994), 9–71.

[10] Arenas-Sánchez, and Dovier, A. Minimal Set Unification. Tr 6/95, Dipartimento di Informatica, Univ. di Pisa, April 1995.

[11] Arenas-Sánchez, P., and Dovier, A. Minimal Set Unification. In *Proc. Seventh Int'l Symp. on Programming Language Implementation and Logic Programming* (1995), M. Hermenegildo and S. D. Swierstra, Eds., vol. 982 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 397–414.

[12] Barbuti, R., Mancarella, P., Pedreschi, D., and Turini, F. A transformation approach to negation in logic programming. *Journal of Logic Programming 8* (1990), 201–228.

[13] Barwise, J., and Moss, L. Hypersets. *The Mathematical Intelligencer 13*, 4 (1991), 31–41.

[14] Barwise, J., and Moss, L. Applying AFA. Notes for a Tutorial on Non-Wellfounded Sets, 1993.

[15] Beeri, C., Naqvi, S., Shmueli, O., and Tsur., S. Set Constructors in a Logic Database Language. *Journal of Logic Programming 10*, 3 (1991), 181–232.

[16] Bellé, D., and Parlamento, F. Undecidability of Weak Membership Theories. In *Proceedings of the International Conference on Logic and Algebra (in memory of R. Magari)* (1994). Siena.

[17] Bernays, P. A system of axiomatic set theory. Part I. *The Journal of symbolic logic 2* (1937), 65–77.

[18] Börger, E., and Rosenzweig, D. The Mathematics of Set Predicates in Prolog. In *Computational Logic and Proof Theory* (1993), D. Mundici, G. Gottlob, and A. Leitsch, Eds., Lecture Notes in Computer Science, Springer-Verlag, Berlin.

[19] Bruscoli, P., Dovier, A., Pontelli, E., and Rossi., G. Compiling Intensional Sets in CLP. In *Proc. Eleventh Int'l Conf. on Logic Programming* (1994), P. Van Entenryck, Ed., The MIT Press, Cambridge, Mass., pp. 647–661.

[20] Burckert, H.-J. *A resolution principle for a logic with restricted quantifiers.* Springer-Verlag, Berlin, 1991.

[21] Bürckert, H.-J., Herold, A., Kapur, D., Siekmann, J. H., Stickel, M. E., Tepp, M., and Zhang, H. Opening the AC-Unification Race. *Journal of Automated Reasoning 4*, 4 (1988), 465–474.

[22] Büttner, W. Unification in the Data Structure Sets. In *Proc. of the Eight International Conference on Automated Deduction* (1986), J. K. Siekmann, Ed., vol. 230, Springer-Verlag, Berlin, pp. 470–488.

[23] Büttner, W., and Simonis, H. Embedding Boolean Expressions into Logic Programming. *Journal of Symbolic Computation 4* (1987), 191–205.

[24] Cantone, D., Ferro, A., and Omodeo., E. G. *Computable Set Theory, Vol. 1.* International Series of Monographs on Computer Science. Clarendon Press, Oxford, 1989.

[25] Chan, D. Constructive Negation Based on the Completed Database. In *Proc. Fifth International Conference and Symposium on Logic Programming* (1988), R. Kowalski and K. Bowen, Eds., The MIT Press, Cambridge, Mass., pp. 111–125.

[26] Chan, D. An Extension of Constructive Negation and its Application in Coroutining. In *Proc. North-American Conference on Logic Programming 89* (1989), E. Lusk and R. Overbeek, Eds., The MIT Press, Cambridge, Mass., pp. 477–493.

[27] Chang, C. C., and Keisler, H. J. *Model Theory.* Studies in Logic. North Holland, Amsterdam, 1973.

[28] Clark, K. L. Negation as Failure. In *Logic and Databases*, H. Gallaire and J. Minker, Eds. Plenum Press, 1978, pp. 293–321.

[29] Cohen, P. J. *Set Theory and the Continuum Hypothesis.* W. A. Benjamin, New York, 1966.

[30] Colmerauer, A. Equations and inequations on finite and infinite trees. In *Proceedings of the* 2^{nd} *Int'l Conf. on Fifth Generation Computer Systems* (1984), pp. 85–99.

[31] Colmeraurer, A. Prolog and Infinite Trees. In *Logic Programming*, K. L. Clark and S.-A. Tarnlund, Eds. Academic Press, New York, 1982, pp. 231–251.

[32] Debray, S., Hermenegildo, M., and Warren, R. Global Flow Analysis as a Practical Compilation Tool. *Journal of Logic Programming 13*, 4 (1992).

[33] Dovier, A. A Language with Finite Sets Embedded in the CLP Scheme. In *Selected papers from* 4^{rd} *Int'l Workshop on Extension of Logic Programming* (1994), R. Dyckhoff, Ed., vol. 798 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin, pp. 77–93.

[34] Dovier, A., Omodeo, E. G., and Policriti, A. Hyperset constraint handling. Rr 21/94, Dipartimento di Matematica ed Informatica, Univ. di Udine, December 1994.

[35] Dovier, A., Omodeo, E. G., Policriti, A., and Rossi, G. Solving Systems of Equations over Hypersets. In *GULP–PRODE'94 1994 Joint Conf. on Declarative Programming* (1994), M. Alpuente, R. Barbuti, and I. Ramos, Eds., pp. 403–417.

[36] Dovier, A., Omodeo, E. G., Pontelli, E., and Rossi, G. {log}: A Language for Programming in Logic with Finite Sets. To appear in the Journal of Logic Programming.

[37] Dovier, A., Omodeo, E. G., Pontelli, E., and Rossi., G. {log}: A Logic Programming Language with Finite Sets. In *Proc.*

Eighth Int'l Conf. on Logic Programming (1991), K. Furukawa, Ed., The MIT Press, Cambridge, Mass., pp. 111–124.

[38] Dovier, A., Omodeo, E. G., Pontelli, E., and Rossi, G. Embedding Finite Sets in a Logic Programming Language. In *Selected papers from* 3^{rd} *Int'l Workshop on Extension of Logic Programming* (1993), E. Lamma and P. Mello, Eds., vol. 660 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin, pp. 150–167.

[39] Dovier, A., and Pontelli., E. La Programmazione Logica con Insiemi. Master's thesis, Università di Udine, 1991. In italian.

[40] Dovier, A., and Pontelli., E. A WAM based Implementation of a Logic Language with Sets. In *Proc. Fifth Int'l Symp. on Programming Language Implementation and Logic Programming* (1993), M. Bruynooghe and J. Penjam, Eds., vol. 714 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 275–290.

[41] Dovier, A., Pontelli, E., and Rossi, G. The CLP language {log}, and the relation between Intensional sets and Negation. NMSU-CSTR-9503, Department of Computer Science, Las Cruces, New Mexico, USA, March 1995.

[42] Dovier, A., and Rossi, G. Embedding Extensional Finite Sets in CLP. In *Proc. of Int'l Logic Programming Symposium, ILPS'93* (1993), D. Miller, Ed., The MIT Press, Cambridge, Mass., pp. 540–556.

[43] Drabent, W. What is Failure? An Approach to Constructive Negation. Tech. rep., Linkoping University, october 1993.

[44] Enderton, H. B. *A mathematical introduction to logic.* Academic Press, 1973. 2^{nd} printing.

[45] Garey, M. R., and Johnson, D. S. *Computers and Intractability – A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[46] Gervet, C. Conjunto: Constraint Logic Programming with Finite Set Domains. In *Proc. of International Logic Programming Symposium, ILPS'94* (1994), M. Bruynooghe, Ed., The MIT Press, Cambridge, Mass., pp. 339–358.

[47] Gogol., D. The $\forall_n\exists$-completeness of Zermelo-Fraenkel set theory. *Zeitschr. f. Logik und Grundlagen d. Math. 24*, 4 (1978), 289–290.

[48] Heintze, N., and Jaffar, J. Set Constraints and Set-Based Analysis. Technical report, Carnegie Mellon University, 1994.

[49] Herbrand, J. Recherches sur la theorie de la demonstration. Master's thesis, Université de Paris, 1930. Also in *Ecrits logiques de Jacques Herbrand*, PUF, Paris, 1968.

[50] Hill, P. M., and Lloyd, J. W. The Gödel report. Tech. Rep. TR-91-02, Computer Science Department, University of Bristol, 1991.

[51] Hill, P. M., and Lloyd, J. W. *The Gödel Programming Language.* The MIT Press, Cambridge, Mass., 1994.

[52] Hopcroft, J. E., and Ullman, J. D. *Introduction to Automata Theory, Languages and Computations.* Addison-Wesley, 1979.

[53] Jaffar, J., and Lassez, J.-L. Constraint Logic Programming. Tech. rep., Department of Computer Science, Monash University, June 1986.

[54] Jaffar, J., and Maher, M. J. Constraint Logic Programming: A Survey. *The Journal of Logic Programming 19–20* (1994), 503–581.

[55] Jayaraman, B. Implementation of Subset-Equational Programs. *Journal of Logic Programming 12*, 4 (1992), 299–324.

[56] Jayaraman, B., and Moon, K. Implementation of Subset Logic Programs. Technical report, 95–14, Department of Computer Science, SUNY Buffalo, March 1995.

[57] Jayaraman, B., and Plaisted, D. A. Programming with Equations, Subsets and Relations. In *Proceedings of NACLP89* (1989), E. Lusk and R. Overbeek, Eds., The MIT Press, Cambridge, Mass., pp. 1051–1068. Cleveland.

[58] Jech, T. J. *Set Theory.* Academic Press, 1978.

[59] Kapur, D., and Narendran, P. NP-completeness of the set unification and matching problems. In *8th International Conference on Automated Deduction* (1986), J. H. Siekmann, Ed., vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 489–495.

[60] Kirk Snyder, W. The SETL2 Programming Language. Research report, Courant Institute of Mathematical Sciences, New York, January 1990.

[61] Knuth, D. E. *The Art of Computer Programming*, vol. 1–Fundamental Algorithms. Addison-Wesley, 1968.

[62] Kozen, D. Logical Aspects of Set Constraints. In *Proceedings 1993 Conference on Computer Science Logic)* (1993), E. Börger, Y. Gurevich, and K. Meinke, Eds., vol. 832 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 175–188.

[63] Kozen, D. Set Constraints and Logic Programming. Technical report 94-1467, Computer Science Department, Cornell University, 1994.

[64] Kunen, K. *Set Theory. An Introduction to Independence Proofs.* Studies in Logic. North Holland, Amsterdam, 1980.

[65] Kuper, G. M. On the Expressive Power of Logic Programming with Sets. In *Proc. 7^{th} ACM SIGMOD Symposium* (1988).

[66] Kuper, G. M. Logic Programming with Sets. *Journal of Computer and System Science 41*, 1 (1990), 66–75.

[67] Lassez, J. L., Maher, M. J., and Marriot, K. Unification revisited. In *Lecture Notes in Computer Science* (1986), vol. 306.

[68] LEGEARD, B., AND LEGROS, E. CLPS: A Set Constraints Logic Programming Language. Tech. rep., Laboratoire d'Automatique de Besançon, Institut de Productique, Besançon, France, February 1991.

[69] LEGEARD, B., AND LEGROS, E. Short overview of the CLPS system. In *Proc. Third Int'l Symposium on Programming Language Implementation and Logic Programming* (august 1991), J. Maluszynsky and M. Wirsing, Eds., vol. 528 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 431–433. Passau, Germany.

[70] LEVY, A. *Basic Set Theory.* Perspectives in Mathematical Logic. Springer Verlag, 1979.

[71] LEWIS, H. R. *Unsolvable Classes of Quantificational Formulas.* Advanced book Program. Addison–Wesley Publishing Company, Inc., 1979.

[72] LIVESEY, M., AND SIEKMANN, J. Unification of Sets and Multisets. Technical report, Institut für Informatik I, Universität Karlsruhe, 1976.

[73] LLOYD, J. W. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second edition.

[74] MAHER, M. J. Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees. In *Proceedings of* 3^{rd} *Symposium Logic in Computer Science* (Edinburgh, 1988), pp. 349–357.

[75] MAL'CEV, A. Axiomatizable Classes of Locally Free Algebras of Various Types. In *The Metamathematics of Algebraic Systems*, Collected Papers. North Holland, Amsterdam, 1971, ch. 23.

[76] MARCHINI, C. Difficoltà del concetto di finito in Teoria degli insiemi. *Cultura e Scuola* (1991).

[77] MARTELLI, A., AND MONTANARI, U. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems 4* (1982), 258–282.

[78] Martelli, A., and Rossi, G. Stepwise Development of an Algorithm for Unification over Infinite Terms. *Computers and Artificial Intelligence 9(3)* (1990), 209–239.

[79] Mendelson, E. *Introduction to Mathematical Logic.* Van Nostrand, Princeton, N. J., 1979.

[80] Montagna, F., and Mancini, A. A minimal predicative set theory. *Notre Dame Journal of Formal Logic 35* (1994), 186–203.

[81] Munakata, T. Notes on implementing sets in prolog. *CACM 35*, 3 (1992).

[82] Naftalin., M. An experiment in practical semantics. In *ESOP86* (1986), vol. 213 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

[83] Naqvi, S., and Tsur., S. *A Logical Language for Data and Knowledge Bases.* Computer Science Press, NY, 1989.

[84] Omodeo, E. G., Parlamento, F., and Policriti., A. Decidability of $\exists^*\forall$-sentences in Membership Theories. Research Report 6, University of Udine, may 1992.

[85] Omodeo, E. G., Parlamento, F., and Policriti, A. A derived algorithm for evaluating ε-expressions over abstract sets. *J. Symbolic Computation 15* (1993).

[86] Omodeo, E. G., and Policriti, A. Solvable set/hyperset contexts: I. some decision procedures for the pure, finite case. *Communication on Pure and Applied Mathematics Special issue dedicated to J. T. Schwartz.* To appear.

[87] Omodeo, E. G., and Policriti, A. Decision procedures for set/hyperset contexts. In *Design and implementation of symbolic computation systems* (1993), vol. 722, Springer-Verlag, Berlin, pp. 192–215.

[88] Omodeo, E. G., Policriti, A., and Rossi., G. F. Che genere di insiemi/multi-insiemi/iperinsiemi incorporare nella programmazione logica? In *Proc. Eighth Italian Conference on Logic Programming* (1993), D. Saccà, Ed., pp. 55–70.

[89] Parlamento, F., and Policriti, A. Decision Procedures for Elementary Sublanguages of Set Theory IX. Unsolvability of the Decision Problem for a Restricted Subclass of δ_0-Formulas in Set Theory. *Communications of Pure and Applied Mathematics 41* (1988), 221–251.

[90] Parlamento, F., and Policriti, A. The logically simplest form of the infinity axiom. *American Mathematical Society 103* (1988).

[91] Parlamento, F., and Policriti, A. Note on "The logically simplest form of the infinity axiom". *American Mathematical Society 108*, 1 (January 1990), 285–286.

[92] Paterson, M. S., and Wegman, M. N. Linear unification. *Journal of Comuter System Science 16*, 2 (1978), 158–167.

[93] Przymusinski, T. On constructive negation in logic programming. Tech. rep., University of Texas in El-Paso, 1989.

[94] Rabin, M. O. Elements of recursion theory. In *Handbook of Mathematical Logic*, J. Barwise, Ed. North Holland, Amsterdam, 1977.

[95] Robinson, J. A. A machine-oriented logic based on the resolution principle. *Journal of the ACM 12* (1965), 23–41.

[96] Schwartz, J. T., Dewar, R. B. K., Dubinsky, E., and Schonberg., E. *Programming with sets, an introduction to SETL.* Springer-Verlag, Berlin, 1986.

[97] Shepherdson, J. C. Negation in Logic Programming. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, 1988, pp. 19–88.

[98] Shepherdson, J. C. Language and equality theory in logic programming. Research Report PM-91-02, University of Bristol, 1991.

[99] Shmueli, O., Tsur, S., and Zaniolo, C. Compilation of Set Terms in the Logic Data Language (LDL). *Journal of Logic Programming 12*, 1 (1992), 89–120.

[100] Shostak, R. E. Deciding Combinations of Theories. In *6th Conference on Automated Deduction* (1982), D. W. Loveland, Ed., vol. 138 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 209–222.

[101] Siekmann, J. H. Unification theory. In *Unification*, C. Kirchner, Ed. Academic Press, 1990.

[102] Spivey., J. M. *The Z Notation: A reference Manual, 2^{nd} edition.* International Series in Computer Science. Prentice Hall, 1992.

[103] Stolzenburg, F. An Algorithm for General Set Unification and its Complexity. In *Unpublished proc. of Workshop on Logic Programming with Sets, in conjunction with ICLP'93* (June 1993), E. G. Omodeo and G. Rossi, Eds. Budapest.

[104] Stuckey, P. J. Negation and Constraint Logic Programming. *Information and Computation 1* (1995), 12–33.

[105] Tarski, A. Sur les ensembles fini. *Fundamenta Mathematicae VI* (1924), 45–95.

[106] Tarski, A., and Givant, S. *A Formalization of Set Theory without Variables*, vol. 41 of *Colloquium Publications.* American Mathematical Society, 1986.

[107] Tarski, A., Mostowski, A., and Robinson, R. M. *Undecidable Theories.* North Holland, Amsterdam, 1953.

[108] Turner, D. An overview of MIRANDA. *SIGPLAN Notices 21*, 12 (1986).

[109] van Heijenoort, J. *From Frege to Gödel, A Source Book in Mathematical Logic, 1879–1931.* Harvard University Press, Cambridge, Massachusetts, London, England, 1977.

[110] Vaught, R. L. On a Theorem of Cobham Concerning Undecidable Theories. In *Proceedings of the 1960 International Concress* (1962), E. Nagel, P. Suppes, and A. Tarski, Eds., Stanford University Press, Stanford, pp. 14–25.