Curriculum Vitae of Marco Comini*

born in Brescia, Italy on Feb 21, 1970

Contents

1 Personal Data

Address	via Tricesimo, 123 – 33100 Udine
Nationality	Italian
Military Service	absolved with civil service on November 7, 1996
Home page	http://www.dimi.uniud.it/comini/
telephone	Tel: +39.043.255.8447 – Fax: +39.043.255.8499
address	Dipartimento di Matematica e Informatica Università Di Udine Via Delle Scienze, 206 33100 Udine Italy

2 Education, Academic Positions and Awards

1989

- **High School**: "Perito Industriale" Diploma *spec. Computer Science*, Istituto Tecnico Industriale Statale *"B.Castelli"*, Brescia, Italy (qualifying exam score 60/60).

*Updated on May 21, 2013

1989–93
- **Master's Candidate: Corso di Laurea in Scienze dell'Informazione** (Computer Science), University of Pisa
- **Enrolled as a Student at *Scuola Normale Superiore di Pisa*** in the "Classe di Scienze" (School of Sciences).

1993
- (5 months) Master's candidates grant by *Consiglio Nazionale delle Ricerche*. (01/09/93–01/02/94)
- **Master's Degree in " Scienze dell'Informazione" (Computer Science)**, University of Pisa on July 16, 1993 (score 110/110 Cum Laude). Master's Thesis Title: A Generalized Semantics for Positive Logic Programs. Supervisor: Prof. G. Levi. Referee: Prof. F. Rossi.
- **Diploma di Licenza (Special Master's Degree) in Scienze dell'Informazione (Computer Science)**, *Scuola Normale Superiore di Pisa*. (01/11/93)

1993–97 (4 years) Ph.D. Student in Computer Science at the *Computer Science Department*, University of Pisa.

1997 **Discussion of the Ph.D. Thesis** to Collegio docenti of the Computer Science Department, University of Pisa. Thesis title: *An abstract interpretation framework for Semantics and Diagnosis of logic programs.* Supervisor Prof. G. Levi; International Referees: Prof. G. Ferrand, Prof. J. Maluszynski; Local Opponents: Prof. R. Barbuti, Prof. G. Gallo.

1998
- (7 months) Guest-researcher at *Institutionen för Datavetenskap, Linköping Universitet, Sweden* (IDA, Linköping University, Sweden). (01/02/98–31/08/98)
- (4 months) Post-Doctoral grant by *Institut National de Recherche en Informatique et en Automatique, Rocquencourt-Paris, France* (INRIA, Rocquencourt-Paris, France). (01/10/98–31/01/99)
- Ph.D. Degree in Computer Science on May 12, 1998. National Judging Commission: Prof. A. De Santis, Prof. S. Martini and Prof. V. Ambriola.

1999-00
- (5 months) Van Vleck Visiting Assistant Professor of Mathematics in the *Department of mathematics, Wesleyan University, Middletown CT, USA*. (01/01/99–31/05/99)
- (2 years) Post-Doctoral research activity grant by *University of Pisa* (Italy), A.Y. 1998-99 and 1999-00,

2000-05 Assistant professor at Università di Udine (University of Udine). Employed since July 3, 2000.

2005-now Associate professor at Università di Udine (University of Udine). Employed since November 2, 2005.

3 Scientific Activities

3.1 The Research Group

I'm coordinating the FLIT research group. We work on development of Sematics-Based Formal Methods to be applied for the realization of (automatic) programming support tools. In particular the topics where we concentrate our efforts are the following:

Analysis, Verification and Correction of Declarative Languages:

- Abstract Diagnosis, Abstract Verification and Analysis of Functional-Logic Languages.
- Abstract Diagnosis, Abstract Verification and Analysis of Timed Concurrent Constraint Languages.
- Abstract Diagnosis, Abstract Verification and Analysis of Logic Languages.
- Type Inference of Functional, Logic and Functional-Logic Languages.

- Verification&Correction of Functional-Logic Languages.

Formal Methods for the Web and Software Engineering:

- Linguaggi di interrogazione e filtering approssimato di documenti semistrutturati.
- Metodi formali per l'analisi e verifica di siti Web.
- UML consistency: Consistency Analysis of UML diagrams.
- UML quality: Quality Analysis of UML diagrams.

Group Members

- Marco Comini (Associate Professor)
- Demis Ballis (Assistant Professor)
- Andrea Baruzzo (Post-Doc)
- Giovanni Bacci (Post-Doc)
- Laura Titolo (PhD student)
- Luca Torella (PhD student)

Scientific Collaborations

The group members collaborate with

- Moreno Falaschi, Michele Baggi. Dipartimento di Scienze Matematiche e Informatiche, Università degli Studi di Siena
- Maria Alpuente, Salvador Lucas, Alicia Villanueva, Santiago Escobar, Daniel Romero. Departamento de Sistemas Informàticos y Computaciòn, Universidad Politécnica de Valencia.
- Ferruccio Damiani. Dipartimento di Informatica, Università di Torino.
- Makoto Tatsuta. National Institute of Informatics, Giappone.

3.2 Description of the Research Activity

Research Topics

- Semantics of Declarative Languages.
- Abstract Interpretation.
- Abstract (Declarative) Diagnosis of Logic Programs.
- Abstract Verification of Logic Programs.
- Abstract Verification/Diagnosis of Functional Programs.
- Abstract Verification/Diagnosis of Functional-Logic Programs.
- Abstract Verification/Diagnosis of Timed Concurrent Constraint Languages.
- Automatic Synthesis of Specifications for Declarative Languages
- Type Inference of Functional, Logic and Functional-Logic Languages.
- Verification of UML diagrams with OCL constraints.
- Quality Analysis of UML diagrams with OCL constraints.

Motivations/Overview

Definite logic programs have a very elegant declarative semantics, i.e., the least Herbrand model. However, some semantics-based techniques (such as program analysis, debugging and transformation) require more traditional semantics which are able to capture computational rather than declarative properties .

Several ad-hoc semantics modeling various abstraction of *SLD*-trees (abstract properties) have been defined, including the ones specifically designed for static program analysis.

The first motivation of this work was to develop a framework to *systematically* derive the semantics modeling an abstract property; to address problems such as the relation between the operational and the denotational semantics and to reason about their properties (e.g. compositionality, correctness and precision degree).

One interesting example of semantics-based technique (concerned with model-theoretic properties) which can take advantage of more concrete semantics is declarative debugging.

Our second motivation was that of applying the results of the theoretical framework to extend sematic-based techniques (declarative debugging, verification and transformation) to cope with the analysis of abstract operational properties described by finite domains, such as groundness dependencies and lot of others.

All my constructions are based on Abstract Interpretation

Abstract interpretation in pills

Abstract interpretation is a theory of the approximation of discrete systems conceived at the end of the 1970s by P. and R. Cousot. This theory allows to formally specify provably correct approximation processes for the behavior of any computing system. For example, abstract interpretation provides methods that are general enough to specify static program analyzers, automatic verifiers of software and hardware systems, automatic verifiers of the properties of communication protocols, type systems and so forth. The abstract interpretation research field is extremely lively: the *International Static Analysis Symposium* (SAS) and the *International Conference on Verification, Model Checking and Abstract Interpretation* (VMCAI) are the annual conferences dedicated to this sector, but numerous contributions are presented also at the *International Conference on Computer Aided Verification* (CAV), *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (POPL) and other conferences/symposia. There is a significant research community, especially in Europe (and in Italy, in particular, as witnessed by numerous PRIN projects on abstract interpretation or related themes), the United States and Asia. Abstract interpretation is having a considerable industrial impact due to the inescapable necessity of guaranteeing the quality of safety critical software. For example, the *PolySpace* static analyzer, which works on C, C++ and Ada programs, was conceived and entirely designed by means of abstract interpretation and it is a product quite successfully commercialized by the multinational company TheMathWorks. The Microsoft Visual Studio development environment includes a module for the static analysis of .NET bytecode for the automatic inference of correctness specifications (the so-called *Code Contracts*). Finally, but this list could continue, we can mention the *ASTRÉE* static analyzer, developed by the research team of Patrick and Radhia Cousot, commercialized by the German company AbsInt, and used by the Airbus consortium for the certification of safety critical, on-board software for the Airbus A340 e A380 airplanes.

Semantics of Logic Languages by Abstract Interpretation [5,2,1,14,13,11] [1]

The idea of using abstract interpretation techniques as unifying framework for various semantics is well-known [CousotC92]. However the originality of our work is that we fully exploit this idea and provide a real flexible framework which provides useful theoretical bases for *new* semantic-based applications.

The first contribution of our work is a uniform framework for the *reconstruction* of existing semantics and for the *systematic* design of *new* semantics able to deal also with the *approximation* typical of static *program analysis*.

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

Using abstract interpretation techniques to model abstraction allows us to state algebraic conditions on several classes of abstract properties w.r.t. the four basic semantic operators. These conditions guarantee the validity of several general theorems, once the correctness of the abstraction has been proven. Depending on the class, we automatically obtain a new denotational semantics, transition system, top-down and bottom-up denotations, together with several interesting theorems (equivalence, compositionality w.r.t. the various syntactic operators, correctness and minimality of the denotations and precision degree). The

[1] For an explanation about CiteSeer.IST - Scientific Literature Digital Library service look at the beginning of Section 3.5.

taxonomy models, within the same framework, both standard semantics than non-standard ones, typical of program analysis. Moreover it permits to reason about its compositionality properties (condensing, OR-compositionality, existence of an abstract transition system).

Our taxonomy is useful to design a *new* semantics with some *a priori* given properties. Since the axioms for a class are sufficient conditions for the derived semantics to have the required properties, one needs only to define an abstract property and check if it belongs to the right class. If it does not, thanks to abstract interpretation composition and refinement, one can compute a more concrete abstract property which does belong to the desired class.

The results of this research have been published in some international conferences [14,13,11], in the PhD Thesis [1], on the journal "Theoretical Computer Science" [2] and on the journal "Information and Computation" [5].

Abstract Diagnosis of (Constraint) Logic Programs [3,33,1,17,32,16,15,30,12]

The results of theoretical framework described in the previous section have been successfully applied to declarative debugging.

Declarative debugging is concerned with model-theoretic properties. The related declarative semantics is the least Herbrand model in [Shapiro82], the set of program completion models in [Lloyd87] and the set of atomic logical consequences in [Ferrand87]. The idea behind declarative debugging is to collect information about what the program is intended to do and compare this with what it actually does. By using these *symptoms*, a diagnoser can find errors.

By applying our framework we have obtained a new diagnosis technique (Abstract Diagnosis) which extends declarative debugging to the case where the intended behavior (specification) of the diagnosed program is finite and defines a program property rather than its semantics. The resulting technique leads to elegant bottom-up and top-down verification methods, which do not require to determine the symptoms in advance, and which are effective in the case of abstract properties described by finite domains.

By investigating on some special instancies we proposed some diagnosis techniques which would be of practical interest.

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

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

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

The theoretical results on partial correctness, completeness and bug derivation are valid for the *diagnosis algorithms* too. I have developed some prototype implementation of the algorithms by means of Prolog meta-programs. Their source code is available on my Home page.

Besides the obtained practical results, the Abstract Diagnosis technique shows that our approach is useful to define new elegant and powerful semantic-based techniques for programming tools.

The results of this research have been published in some international conferences [17,16,15,12], in the PhD Thesis [1], on the "Journal of Logic Programming" [3].

Within the european Esprit DiSCiPL project the same approach has been applied to Constraint Logic Languages. Some results of this research have been published in an international workshop [33].

Abstract Verification of (Constraint) Logic Programs [7,4,20,6,19,18]

During the development of Abstract Diagnosis we noted that some of its characteristics were typical of program verification. Thus we have applied the concepts of the theoretical framework to various problems related to the verification of logic programs.

The aim of verification is to define conditions which allow us to formally prove that a program behaves as expected, i.e., that the program is correct w.r.t. a given specification, a description of the program's expected behavior. We have successfully used the framework (and abstract interpretation) techniques to organize and synthesize proof methods for program verification. Verification techniques inherit the nice features of abstract interpretation. Namely, the resulting verification framework (Abstract Verification framework) is parametric with respect to the (abstract) property we want to model. Given a specific property, the corresponding verification conditions are systematically derived from the framework and guaranteed to be indeed sufficient partial correctness conditions. By choosing a suitable domain, which leads to finite specifications, these sufficient conditions are effectively computable.

I have developed, as a case study, a prototype implementation of the verification algorithms on a type domain [20]. Its source code is available on my Home page.

We have shown that within the framework we can reconstruct well-known methods, using extensional semantics w.r.t. pre-post conditions, such as success-correctness [Clark79, Deransart93], I/O correctness [Drabent97] and I/O and call correctness [DrabentM88, BossiC89, AptM94].

The verification framework can be instantiated to specifications given in terms of assertions (which can be viewed as an intensional semantics). We have shown that assertions can indeed be handled as abstract domains and have shown two applications with different specification languages.

The first one is a simple decidable assertion language, which is able to express properties of terms, including types and other properties relevant to static analysis. An open interesting issue is the definition of more expressive (still decidable) specification languages.

The second one allows the user to specify properties to be used in the assertions by means of CLP programs. We have shown, through some examples, how the resulting sufficient verification conditions can be derived and proved by using program transformations techniques. Most of the verification conditions can very easily be proven by using a few unfolding steps, while other transformation techniques, such as goal replacement, are needed to prove more complex properties. As we have shown in the examples, the generation of the intermediate lemmata needed for goal replacement can often be obtained by using an unfold/fold proof method, as stated in [PettorossiP99]. Our examples together with these considerations suggest that the process of proving verification conditions can easily be semi-automatized by using, for example, the tool MAP [PettorossiP99] as we showed in our examples.

The results of this research have been published in some international conferences [20,6,19,18] and are illustrated in detail in a paper published on journal "Science of Computer Programming" [7].

Verification/Diagnosis of Functional Programs [39,9,21]

The approach to Semantics, Diagnosis and Verification of Logic Programs can be generalized to other paradigms. We just need to define a fixpoint semantics on the concrete domain. The compositionality properties will be of course different and related to the language syntactic operators. Recently we have obtained some promising results for Functional Programming.

In [21] we have obtained a generic scheme for the declarative debugging of functional programs modeled as term rewriting systems. We can use (depending on the needs) two different concrete semantics which models the computed values or the normal forms. With these semantics we developed a finitely terminating bottom-up diagnosis method, which can be used statically. Our debugging framework does not require the user to either provide error symptoms in advance or answer questions concerning program correctness. We have considered, as a case study, an instance of the method over the $depth(k)$ domain and made available a prototypical implementation in Haskell which has been tested on some non trivial examples.

This work is motivated by the fact that a "compact" semantics for term rewriting systems, which is essential for the development of effective semantics-based program manipulation tools (e.g. automatic program analyzers and debuggers), does not exist.

Thus in [9] we have developed such a "compact" semantics for term rewriting systems. The big-step rewriting semantics that is most commonly considered in functional programming is the set of values/normal forms that the program is able to compute for any input expression. Such a big-step semantics is unnecessarily oversized, as it contains many "semantically useless" elements that can be retrieved from a smaller set of terms. Therefore, in this article, we present a compressed, goal-independent collecting fixpoint semantics that contains the smallest set of terms that are sufficient to describe, by semantic closure, all possible rewritings. The compactness of the semantics makes it suitable for applications. Actually, our semantics can be finite whereas the big-step semantics is generally not, and even when both semantics are infinite, the fixpoint computation of our semantics produces fewer elements at each step. To support this claim we report several experiments performed with a prototypical implementation (URL `http://safe-tools.dsic.upv.es/zipit`)

It is interesting to note that the resulting methodology can be applied to a class of Term Rewriting Systems much bigger than previous approaches (like Echahed and Hanus).

The results of this research have been published in an international conference [21] on the journal "Theoretical Computer Science" [9].

Given the interest of the scientific community and the still open possibilities of this research topic I'm going to deeply develop this framework in the future.

39

Verification/Diagnosis of Functional-Logic Programs [37,26]

As already said, the approach to Semantics, Diagnosis and Verification of Logic Programs can be can be generalized to other paradigms. We are now working on an extension of the framework to cope with (integrated) functional-logic languages.

Given the absence in literature of previous candidates to work on, in [37] we opted for the systematic use of Abstract Interpretation. Thus

1. we defined, for the first order fragment of Curry, a (very) concrete semantics which models completely *needed narrowing* derivations;
2. then, we have defined the abstraction for computed answers and derived "automatically" its semantics, which turned out to be imprecise. Thus we have successively refined such an abstraction till we have obtained a precise refinement.

Thanks to this semantics in [26] we could apply the Abstract Diagnosis methodology.

Verification/Diagnosis of Functional-Logic Programs [38,10, 41]

We are now working on an extension of the framework to cope with timed concurrent constraint logic languages.

Given the absence in literature of previous candidates to work on, we have started to develop one in [38].

Thanks to this semantics we could apply the Abstract Diagnosis methodology in [10,41].

Automatic Synthesis of Specifications for Declarative Languages [40,29,27,28]

Type-Inference via Abstract Interpretation [25]

The results of this research have been published in an international conference [25] .

Verification/Quality of UML diagrams with OCL constraints [23,8,35,36,24,22]

With the advent of Model-Driven Development, models has become a central component in the software development process. As code can be (semi)automatically generated from models, the quality of models has a direct impact on the quality of the final product. Furthermore, detecting defects at the model level allows correcting them early in the development process, when it is easier and cheaper.

Formal methods provide many approaches and tools with mathematically sound foundations that can be used to automatically assess model quality. However, there are several barriers in the adoption of formal methods within the software engineering community (amongst others, the mathematics expertise required to software practitioner in order to apply them effectively).

Thus, the main goal of this thesis is the development of *light-weight formal methods* that can be used to assess the quality of a UML model by means of automatic model analysis.

Two different methods are presented in this work:

1. Model Verification for Consistency
2. Model Verification for Quality Critiquing

The first one is a preliminary step in a more ambitious direction of *Model Verification for Correctness* and it is aimed to identify inconsistences between different diagrams in the model. This method is an attempt to analyze UML diagrams and OCL specifications in order to discover errors (i.e. assertion violations) or warnings (i.e. incomplete specifications) in the model or possibly in the assertions themselves.

The second method is aimed to analyze a UML model in order to automatically check if it embodies software design best practices (e.g., design patterns) by means of design critiques.

Common to both the verification approaches discussed here is the observation that recovering from faulty reasoning *after* the time in which the fault is firstly introduced is a very common, but tedious and expensive practice. Therefore, the spirit of this thesis is to develop suitable verification methods which should aid the designer to catch those critical aspects of a model as early as possible, and in a possibly non-obtrusive way.

In order to achieve these results, the aforementioned methods are presented in the context of a suitable methodology which describes how they can be integrated in a typical Model-Driven Development environment.

The results of this research have been published in some international conferences [23,8,22].

3.3 Research Projects

European

1998 Member of the ESPRIT (EU information technologies programme) project "Debugging systems for constraint programming"

2004–2005 Member of the EU project "ICT for EU-India Cross-Cultural Dissemination" (grant ALA/95/23/2003/077-054)

National

1998–2000
- Member of the MURST co-financed project "Sistemi formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software" (Formal Systems for the specification, analysis, verification, synthesis and transformation of software systems). MURST is the Ministry for the Scientific and Technologic Research. [coordinator G. Levi]
- Member of the CNR co-financed project "Verifica, analisi e trasformazione di programmi logici" (Verification, analysis and transformation of Logic Programs). CNR is the National Council for Research. [coordinator G. Levi]

2000–2001 Member of the MURST co-financed project "Certificazione Automatica di Programmi mediante Interpretazione Astratta" (Automatic program certification by abstract interpretation). [coordinator R. Giacobazzi]

2001–2002 Member of the MURST co-financed project "Interpretazione astratta, sistemi di tipo e analisi Control-Flow" (Abstract Interpretation, type systems and control-flow analysis). [coordinator G. Levi]

2004–2005 Member of the MURST co-financed project "Rappresentazione e gestione di dati spaziali e geografici in WEB" (WEB-based management and representation of spatial and geographic data) [coordinator E. Bertino]

2005–2006 Member of the MURST co-financed project "Interpretazione Astratta: Sviluppo e Applicazioni" (AIDA - Abstract Interpretation: Design and Applications) [coordinator R. Giacobazzi]

2007–2008 Member of the MURST co-financed project "Sistemi e calcoli di ispirazione biologica e loro applicazioni" (BISCA - Bio-Inspired Systems and Calculi with Applications) [coordinator P. Degano]

Regional

2002–2003 Member of the Regione FVG co-financed project "Verifica formale, certificazione e model checking per sistemi reattivi, concorrenti ed embedded" (Formal verification, certification and Model-Checking for reactive, concurrent and embedded systems). Regione FVG is the Friuli Venezia Giulia Regional Council. [coordinator A. Policriti]

3.4 PhD. Thesis Supervision

2005-08 "A Unified Framework for Automated UML Model Analysis", Dipartimento di matematica e Informatica, Università di Udine (Andrea Baruzzo).

2009-12 "Abstract Diagnosis and Verification of Functional and Functional-Logic Programs", Dipartimento di matematica e Informatica, Università di Udine (Giovanni Bacci).

2010- "An Abstract Interpretation Framework for Semantics and Diagnosis of Term Rewriting Systems, Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche, Università di Siena (Luca Torella).

2011- "An Abstract Interpretation Framework for Semantics and Verification of Timed Concurrent Constraint Languages, Dipartimento di matematica e Informatica, Università di Udine (Laura Titolo).

3.5 Publications

Ph.D. Thesis

1. M. Comini. *An abstract interpretation framework for Semantics and Diagnosis of logic programs.* Ph.D. thesis TD-5/98, Dipartimento di Informatica, Università di Pisa, 1998.

Journals (ISI Science Citation Index, with referee)

2. M. Comini and M. C. Meo. Compositionality properties of *SLD*-derivations. *Theoretical Computer Science*, 211(1-2):275–309, 1999.
3. M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Abstract diagnosis. *Journal of Logic Programming*, 39(1-3):43–93, 1999.
4. M. Comini, R. Gori, G. Levi, and P. Volpe. Abstract Interpretation based Verification of Logic Programs. *Electronic Notes in Theoretical Computer Science*, 30:1–17, 1999.
5. M. Comini, G. Levi, and M. C. Meo. A Theory of Observables for Logic Programs. *Information and Computation*, 169:23–80, 2001.
6. M. Comini, R. Gori, and G. Levi. Logic programs as specifications in the inductive verification of logic programs. *Electronic Notes in Theoretical Computer Science*, 48:1–16, 2001.
7. M. Comini, R. Gori, G. Levi, and P. Volpe. Abstract Interpretation based Verification of Logic Programs. *Science of Computer Programming*, 49(1–3):89–123, 2003.

8. D. Ballis, A. Baruzzo, and M. Comini. A rule-based method to match Software Patterns against UML Models. *Electronic Notes in Theoretical Computer Science*, 219:51–66, 2007.
9. M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and J. Iborra. A Compact Fixpoint Semantics for Term Rewriting Systems. *Theoretical Computer Science*, 411(37):3348–3371, 2010.
10. M. Comini, L. Titolo, and A. Villanueva. Abstract Diagnosis for Timed Concurrent Constraint programs. *Theory and Practice of Logic Programming*, 11(4-5):487–502, 2011.

International Conference/Workshop Proceedings (with 3 referees at least)

11. M. Comini and G. Levi. An algebraic theory of observables. In M. Bruynooghe, editor, *Proceedings of the 1994 International Symposium on Logic Programming*, pages 172–186. The MIT Press, 1994.
12. M. Comini, G. Levi, and G. Vitiello. Abstract debugging of logic programs. In L. Fribourg and F. Turini, editors, *Proceedings Logic Program Synthesis and Transformation and Meta-programming in Logic 1994*, volume 883 of *Lecture Notes in Computer Science*, pages 440–450. Springer-Verlag, 1994.
13. M. Comini and G. Levi. Beyond the *s*-semantics: a theory of observables. In A. Ursini and P. Aglianò, editors, *Logic and Algebra*, volume 180 of *Lecture Notes in Pure and Applied Mathematics*, pages 25–67. Marcel Dekker, Incorporated, New York, 1995.
14. M. Comini, G. Levi, and M. C. Meo. Compositionality of *SLD*-derivations and their abstractions. In J. Lloyd, editor, *Proceedings of the 1995 International Symposium on Logic Programming*, pages 561–575. The MIT Press, 1995.
15. M. Comini, G. Levi, and G. Vitiello. Declarative diagnosis revisited. In J. Lloyd, editor, *Proceedings of the 1995 International Symposium on Logic Programming*, pages 275–287. The MIT Press, 1995.
16. M. Comini, G. Levi, and G. Vitiello. Efficient Detection of Incompleteness Errors in the Abstract Debugging of Logic Programs. In M. Ducassé, editor, *Proc. 2nd International Workshop on Automated and Algorithmic Debugging, AADEBUG'95*, pages 1–17, 1995.
17. M. Comini, G. Levi, M. C. Meo, and G. Vitiello. Proving properties of logic programs by abstract diagnosis. In M. Dams, editor, *Analysis and Verification of Multiple-Agent Languages, 5th LOMAPS Workshop*, number 1192 in Lecture Notes in Computer Science, pages 22–50. Springer-Verlag, 1996.
18. M. Comini, R. Gori, and G. Levi. Assertion based Inductive Verification Methods for Logic Programs. In A. K. Seda, editor, *Proceedings of MFCSIT'2000*, volume 40 of *Electronic Notes in Theoretical Computer Science*, pages 1–18. Elsevier Science Publishers, 2001. [2]
19. M. Comini, R. Gori, and G. Levi. How to Transform an Analyzer into a Verifier. In R. Nieuwenhuis and A. Voronkov, editors, *Proceedings LPAR 2001*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 595–609. Springer-Verlag, 2001.
20. M. Comini. VeriPolyTypes: a tool for Verification of Logic Programs with respect to Type Specifications. In M. Falaschi, editor, *Proceedings of 11th International Workshop on Functional and (constraint) Logic Programming*, number UDMI/18/2002/RR in Research Reports, pages 233–236, Udine, Italy, 2002. Dipartimento di Matematica e Informatica, Università di Udine.
21. M. Alpuente, M. Comini, S. Escobar, M. Falaschi, and S. Lucas. Abstract Diagnosis of Functional Programs. In M. Leuschel, editor, *Logic Based Program Synthesis and Transformation – 12th International Workshop, LOPSTR 2002, Revised Selected Papers*, volume 2664 of *Lecture Notes in Computer Science*, pages 1–16, Berlin, 2003. Springer-Verlag.
22. A. Baruzzo and M. Comini. Static Verification of UML Model Consistency. In D. Hearnden, J. G. Süß, B. Baudry, and N. Rapin, editors, *MoDeV²a: Model Development, Validation and Verification*. University of Queensland, Le Commissariat à

[2] Available at URL: http://www.elsevier.nl/locate/entcs/volume40.html

l'Energie Atomique - CEA, October 2006.
23. D. Ballis, A. Baruzzo, and M. Comini. A Minimalist Visual Notation for Design Patterns and Antipatterns. In *5th International Conference on Information Technology: New Generations*, pages 51–56. IEEE Computer Society, 2008.
24. A. Baruzzo and M. Comini. A Methodology for UML Models V&V. In *Proceedings of First International Conference on Software Testing, Verification, and Validation*, pages 513–516. IEEE Computer Society, 2008.
25. M. Comini, F. Damiani, and S. Vrech. On Polymorphic Recursion, Type Systems, and Abstract Interpretation. In M. Alpuente and G. Vidal, editors, *Static Analysis – 15th International Symposium, SAS 2008*, volume 5079 of *Lecture Notes in Computer Science*, pages 144–158, Berlin, 2008. Springer-Verlag.
26. G. Bacci and M. Comini. Abstract Diagnosis of First Order Functional Logic Programs. In M. Alpuente, editor, *Logic-based Program Synthesis and Transformation, 20th International Symposium*, volume 6564 of *Lecture Notes in Computer Science*, pages 215–233, Berlin, 2011. Springer-Verlag.
27. G. Bacci, M. Comini, M. A. Feliú, and A. Villanueva. The additional difficulties for the automatic synthesis of specifications posed by logic features in functional-logic languages. In A. Dovier and V. S. Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*, volume 17 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 144–153, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum Fuer Informatik.
28. G. Bacci, M. Comini, M. A. Feliú, and A. Villanueva. Automatic Synthesis of Specifications for First Order Curry Programs. In *Proceedings of the 14th symposium on Principles and practice of declarative programming*, pages 25–34, New York, NY, USA, 2012. ACM.
29. M. Comini and L. Torella. TRSynth: a Tool for Automatic Inference of Term Equivalence in Left-linear Term Rewriting Systems. In *ACM SIGPLAN 2013 Workshop on Partial Evaluation and Program Manipulation (PEPM'13)*. ACM, 2013. To appear.

National Conference/Workshop Proceedings

30. M. Comini, G. Levi, and G. Vitiello. On the Abstract Diagnosis of Logic Programs. In M.I. Sessa, editor, *Proceedings GULP-PRODE'95*, pages 41–57, 1995.
31. R. Bagnara, M. Comini, F. Scozzari, and E. Zaffanella. The AND-compositionality of CLP computed answer constraints. In M. Navarro, editor, *Proceedings of the APPIA-GULP-PRODE'96 Joint Conference on Declarative Programming*, pages 355–366, 1996.
32. M. Comini, G. Levi, and G. Vitiello. Modular abstract diagnosis. In J. L. Freire and M. Falaschi, editors, *Proceedings of the APPIA-GULP-PRODE'98 Joint Conference on Declarative Programming*, pages 409–420, 1998.
33. M. Comini, W. Drabent, and P. Pietrzak. Diagnosis of CHIP programs using type information. In M. C. Meo and M. Vilares Ferro, editors, *Appia-Gulp-Prode'99, Joint Conference on Declarative Programming*, pages 337–349, L'Aquila, Italy, 1999.
34. A. Baruzzo and M. Comini. Checking UML Model Consistency. In *Proceedings of CILC 2006 - Convegno Italiano di Logica Computazionale*, 2006.

Submitted papers

35. A. Baruzzo and M. Comini. A Framework for Computer Aided Consistency Verification of UML Models. [3]
36. A. Baruzzo and M. Comini. Toward a Unified Framework for Quality and Consistency Verification of UML Models. [4]
37. G. Bacci and M. Comini. A Fully-Abstract Condensed Goal-Independent Bottom-Up Fixpoint Modeling of the Behaviour of First Order Curry. Technical Report DIMI-UD/06/2010/RR, Dipartimento di Matematica e Informatica, Università di Udine,

[3] Available at URL: http://www.dimi.uniud.it/comini/Papers/FramCAConVerUML/FramCAConVerUML.pdf
[4] Available at URL: http://www.dimi.uniud.it/comini/Papers/FrameworkQualConsUML/FrameworkQualConsUML.pdf

2010. [5].

38. M. Comini, L. Titolo, and A. Villanueva. A Condensed Goal-Independent Bottom-Up Fixpoint Modeling the Behavior of tccp. Technical report, DSIC, Universitat Politècnica de València, 2013. URL: http://riunet.upv.es/handle/10251/8351.
39. M. Comini and L. Torella. A Condensed Goal-Independent Fixpoint Semantics Modeling the Small-Step Behavior of Rewriting. Technical Report DIMI-UD/01/2013/RR, Dipartimento di Matematica e Informatica, Università di Udine, 2013. URL: http://www.dimi.uniud.it/comini/Papers/.
40. M. Comini and L. Torella. Automatic Inference of Term Equivalence in Term Rewriting Systems. Technical Report DIMI-UD/02/2013/RR, Dipartimento di Matematica e Informatica, Università di Udine, 2013. URL: http://www.dimi.uniud.it/comini/Papers/.
41. M. Comini, L. Titolo, and A. Villanueva. Abstract Diagnosis for tccp using a Linear Temporal Logic. URL: http://www.dimi.uniud.it/comini/Papers/.

Supervied PhD thesis

42. A. Baruzzo. *A Unified Framework for Automated UML Model Analysis*. PhD thesis, Dipartimento di matematica e Informatica, 2008.
43. G. Bacci. *An Abstract Interpretation Framework for Semantics and Diagnosis of Lazy Functional-Logic Languages*. PhD thesis, Dipartimento di matematica e Informatica, 2011.

3.6 Conference Committees

1. **Chairman of the program committee** International Workshop Tools and Environments for (Constraint) Logic Programming. Post-conference workshop International Symposium on Logic Programming, Port Jefferson, NY (USA), October 1997.
2. **AGP01 Program Committee member of** Joint International Conference Appia-Gulp-Prode'01, Evora (PT), Settembre 2001.
3. **WFLP02 Chairman organizing committee** 11th International Workshop Functional and (Constraint) Logic Programming, Grado, Giugno 2002.
4. **WFLP02 Guest Editor** of *Selected Papers from WFLP'02 – 11th International Workshop on Functional and (Constraint) Logic Programming*, volume 76 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 2002. Available at URL: http://www.elsevier.nl/locate/entcs/volume76.html.
5. **WFLP08 Program Committee member of** 17th Int'l Workshop on Functional and (Constraint) Logic Programming, Siena (Italy), July, 2008.
6. **VALID09 Program Committee member of** First International Conference on Advances in System Testing and Validation Lifecycle, Porto (Portugal), September, 2009.
7. **LOPSTR10 Program Committee member of** 20th International Symposium on Logic-Based Program Synthesis and Transformation, Hagenberg (Austria), July, 2010.
8. **VALID10 Program Committee member of** Second International Conference on Advances in System Testing and Validation Lifecycle, Nice (France), August, 2010.
9. **TAP11 Program Committee member of** 5th International Conference on Tests and Proofs, Zurich (Switzerland), June, 2011.
10. **TAP12 Program Committee member of** 6th International Conference on Tests and Proofs, Prague (Czech Republic), May, 2012.
11. **TAP13 Program Committee member of** 7th International Conference on Tests and Proofs, Budapest (Hungary), June, 2013.

[5] Available at URL: http://www.dimi.uniud.it/comini/Papers/

3.7 Invited Talks

1. Una caratterizzazione algebrica degli Osservabili. *Dipartimento di Informatica, Università di Salerno*, Jan 20, 1995.
2. An abstract interpretation framework for semantics of logic programs. *IDA, Linköping Universitet, Sweden*, April 24, 1998.
3. Abstract Diagnosis, an abstract interpretation framework for diagnosis of logic programs. *IDA, Linköping Universitet, Sweden*, May 20, 1998.
4. The approach of Abstract Diagnosis and its recent extensions to type diagnosis of CHIP programs. *Rocquencourt research unit, INRIA, France*, June 23, 1998.
5. (Cycle of 5 seminars) Abstract Diagnosis. *LIFO project, Université d'Orleans, France*, November'98-January'99.
6. An abstract interpretation framework for semantics of logic programs. *Sèminaire "Semantique et Interpretation Abstraite", Ecole Normale Superieure, Paris, France*, November 20, 1998.
7. An abstract interpretation framework for abstract diagnosis of logic programs. *Sèminaire "Semantique et Interpretation Abstraite", Ecole Normale Superieure, Paris, France*, November 27, 1998.
8. Abstract Diagnosis. *Université d'Orleans, France*, December 14, 1998.
9. Abstract Interpretation based Verification and Diagnosis of Logic Programs. *Universidad Politecnica de Valencia, Spain*, May 15, 2002.
10. Applications of Abstract Verification. *Universidad Politecnica de Valencia, Spain*, May 22, 2002.
11. A Static Approach to Consistency Verification of UML Models. *Universidad Politecnica de Valencia, Spain*, June 1, 2006.

3.8 Referee Activities

Journals

1. TCS (2008,2010)
2. TPLP (2000-2001,2003,2004)

Conferences

1. APPIA-GULP-PRODE'96, AGP'01, AGP'02
2. ESOP'99
3. FLOPS'02
4. ICLP'99, ICLP'02, ICLP'03, ICLP'08
5. ICTCS'01, ICTCS'07
6. LICS'02
7. LOPSTR'10
8. LPAR'06
9. PEPM'08
10. PLILP/ALP'98
11. PPDP'03, PPDP'04, PPDP'08
12. SAC'05
13. TAP'11, TAP'12, TAP'13
14. TECLP'97
15. VALID'09, VALID'10
16. WFLP'01, WFLP'02, WFLP'07, WFLP'08
17. WWV'05

3.9 Conference Session Chairman

1. Joint International Conference Appia-Gulp-Prode'00, La Havana (Cuba), December 2000.

2. International Workshop Functional and (Constraint) Logic Programming, Grado (I), June 2002.
3. Joint International Conference Appia-GULP-PRODE 2002, Madrid (Spain), September 2002.
4. 17th Int'l Workshop on Functional and (Constraint) Logic Programming, Siena (Italy), July, 2008.

3.10 Conference Talks

1. "A Generalized Semantic Framework for CLP", Workshop progetto nazionale "Modelli della Computazione e dei Linguaggi di Programmazione", Volterra (I), September 1993.
2. "An Algebraic Theory of Observables", International Symposium on Logic Programming, Ithaca (USA), November 1994.
3. "Compositionality of *SLD*-derivations", International workshop Abstract Interpretation of Logic Languages, Eilat (Israel), June 1995.
4. "Compositionality of *SLD*-derivations and their abstractions", GULP-PRODE'95, Vietri (I), September 1995.
5. "Approximated Abstract Compositional Semantics", International Workshop on Abstract Interpretation of Logic Languages (WAILL'96 I), Pisa (I), February 1996.
6. "Proving Properties of Logic Programs by Abstract Diagnosis", International Workshop on Abstract Interpretation of Logic Languages (WAILL'96 II), Jerusalem (Israel), December 1996.
7. "Diagnosis of CHIP programs using type information", International workshop Types for Constraint Logic Programming, Manchester (UK), June 1998.
8. "Diagnosis of CHIP programs using type information", Joint International Conference Appia-Gulp-Prode'99, L'Aquila (I), September 1999.
9. "Design of Verification Methods by Abstract Interpretation", Concluding Workshop of the Project "Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software." Venezia (I), January 2000.
10. "Logic programs as specifications in the inductive verification of Logic Programs", Joint International Conference Appia-Gulp-Prode'00, La Havana (Cuba), December 2000.
11. "Abstract Interpretation based Verification Methods", Workshop of the Project "Certificazione Automatica di Programmi mediante Interpretazione Astratta" Venezia (I), February 2001.
12. "How to transform an analyzer into a verifier", Joint International Conference Appia-Gulp-Prode'01, Evora (PT), September 2001.
13. "VeriPolyTypes: a tool for Verification of Logic Programs w.r.t. Type Specifications", International Workshop Functional and (Constraint) Logic Programming WFLP'02, Grado (I), June 2002.
14. "The perspective of the Udine's Research Group within the AIDA project", Aida Febbraio 2005, Verona.
15. "An Effective Fixpoint Semantics for General Term Rewriting Systems", Aida 2005, Pisa.
16. "A Static Approach to Consistency Verification of UML Models", Aida 2006, Venezia.
17. "A Static Approach to Consistency Verification of UML Models", CILC 2006, Bari.
18. "On Polymorphic Recursion, Type Systems, and Abstract Interpretation", SAS 2008, Valencia.

3.11 Conference Participation

1. IV Convegno Nazionale di Informatica Teorica, L'Aquila (I), October 1992.
2. EQUADIFF 8: Czecho-Slovak Conference on Differential Equations and their Applications, Bratislava (Slovakia), August 1993.
3. Workshop progetto nazionale "Modelli della Computazione e dei Linguaggi di Programmazione", Volterra (I), September 1993.
4. International Conference on Logic Programming, S. Margherita Ligure (I), July 1994.

5. International Symposium on Logic Programming, Ithaca (USA), November 1994.
6. International workshop Abstract Interpretation of Logic Languages, Eilat (Israel), June 1995.
7. GULP-PRODE'95, Vietri (I), September 1995.
8. International Workshop on Abstract Interpretation of Logic Languages (WAILL'96 I), Pisa (I), February 1996.
9. International Workshop on Abstract Interpretation of Logic Languages (WAILL'96 II), Jerusalem (Israel), December 1996.
10. APPIA-GULP-PRODE'97, Grado (I), June 1997.
11. International Workshop on Abstract Interpretation of Logic Languages (WAILL'97), Grado (I), June 1997.
12. Joint International Conference and Symposium on Logic Programming (JICSLP'98), Manchester (UK), June 1998.
13. International workshop Types for Constraint Logic Programming, Manchester (UK), June 1998.
14. Joint International Symposia SAS and PLILP/ALP'98, Pisa (I), September 1998.
15. Joint International Conference Appia-Gulp-Prode'99, L'Aquila (I), September 1999.
16. Concluding Workshop of the Project "Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software." Venezia (I), January 2000.
17. Joint International Conference Appia-Gulp-Prode'00, La Havana (Cuba), December 2000.
18. Workshop of the Project "Certificazione Automatica di Programmi mediante Interpretazione Astratta" Venezia (I), February 2001.
19. International Workshop Functional and (Constraint) Logic Programming, Kiel (D), September 2001.
20. Joint International Conference Appia-Gulp-Prode'01, Evora (PT), September 2001.
21. International Workshop Functional and (Constraint) Logic Programming WFLP'02, Grado (I), June 2002.
22. Joint International Conference SAS-LOPSTR-AGP 2002, Madrid (Spain), September 2002.
23. Workshop "Giornata GULP 2002". Bologna (I), October 2002.
24. Workshop Progetto Cofinanziato "COVER" Bologna (I), February 2003.
25. International Workshop Functional and (Constraint) Logic Programming WFLP'03, Valencia (S), June 2003.
26. "Convegno Italiano di Logica Computazionale" CILC04, Parma (I), June 2004.
27. Aida Febbraio 2005, Verona.
28. Aida 2005, Pisa.
29. Aida 2006, Venezia.
30. CILC 2006, Bari.
31. WFLP 2008, Siena.
32. LOPSTR 2008, Valencia.
33. SAS 2008, Valencia.
34. LOPSTR 2010, Hagenberg.

3.12 Local Scientific Activities

1. A. Baruzzo, M. Comini, Seminars on UML and OCL, Università degli Studi di Udine, Italy, June-July 2006.
   M. Comini:
   (a) Introduzione ad OCL. July 6, 2006.
   (b) Caratterisitiche di OCL. July 14, 2006.
   (c) Invarianti, Collections, Pre/Post-condizioni. July 20, 2006.

   A. Baruzzo:
   (a) Diagrammi di Struttura. May 30, 2006.
   (b) Diagrammi di Sequenza. June 13, 2006.
   (c) Diagrammi di Attività/Casi d'Uso. June 20, 2006.

(d) Diagrammi di Stato. June 4, 2006.
(e) Caso di Studio I. June 11, 2006.
(f) Caso di Studio II. June 18, 2006.

4 Academic/Institutional Duties

1. Rappresentante Associati in Giunta di Dipartimento del Dipartimento di Matematica e Informatica di Udine dal Novembre 2007 ad oggi.

2. Componente del Consiglio di Classe della Scuola Superiore, dal Settembre 2007 ad oggi.

3. Coordinatore Borse SOCRATES/ERASMUS con *Universidad Politecnica de Valencia* dal 2005 ad oggi.

4. Coordinatore Borse SOCRATES/ERASMUS con *Universidad de Castilla la Mancha* dal 2005 ad oggi.

5. Membro designato (dal Consiglio di Corso di Laurea) della Commissione Piani di Studio dei Corsi di Laurea in Informatica della Facoltà di Scienze, Università di Udine dall'Ottobre 2004 ad oggi.

6. Membro Rappresentante designato (dalla Facoltà di Scienze) nella Commissione Permanente delle Facoltà dell'Università di Udine dal 2004 ad oggi.

7. Membro del Collegio Docenti del Dottorato in Informatica del Dipartimento di Matematica e Informatica di Udine dal Novembre 2005 ad oggi.

8. Responsabile d'Ateneo del Centro di Competenza Universitario del progetto EUCIP4U (coord. Facoltà di Scienze e Facoltà di Ingegneria), dal Febbraio 2006 ad oggi.

9. Commissario concorso d'ammissione alla Scuola Superiore dell'Università di Udine, Settembre 2008.

10. Commissario concorso d'ammissione alla Scuola Superiore dell'Università di Udine, Settembre 2007.

11. Commissario concorso d'ammissione alla Scuola Superiore dell'Università di Udine, Settembre 2006.

12. Commissario concorso d'ammissione alla Scuola Superiore dell'Università di Udine, Settembre 2005.

13. Commissario a Procedura di Valutazione Comparativa a un posto di Ricercatore INF/01, facoltà di Medicina e Chirurgia, Università di Torino, Ottobre-Dicembre 2005.

14. Rappresentante eletto dei ricercatori nel Consiglio di Facoltà di Scienze dall'Ottobre 2003 all'Ottobre 2005.

15. Rappresentante dei Ricercatori in Giunta di Dipartimento del Dipartimento di Matematica e Informatica di Udine dal 13 Novembre 2002 al 31 Ottobre 2005.

16. secretary of the national association for Logic Programming (Gruppo Utenti Logic Programming – GULP) from june 2000 to 2005.

17. Rappresentante della Facoltà di Scienze Matematiche Fisiche e Naturali nella Commissione Permanente delle Facoltà dell'Università di Udine dal Dicembre 2004.

18. Membro della commissione Gestione Spazi Dipartimentali del Dipartimento di Matematica e Informatica di Udine negli anni 2001-2003.

19. web master, since year 2000, of the Web Site of the GULP association (http://www.dimi.uniud.it/gulp/)

20. web master, since year 2002, of the Web Site of the 11th International Workshop on Functional and (constraint) Logic Programming (`http://www.dimi.uniud.it/~wflp2002/`)

21. during years 2001-2003 manager of the scientific publication departmental database.

5 Implementation Projects

Microprocessor control of a robotic arm. The Z80 system I realized can acquire symbolic information (via a RS-232 serial line) about the intended 3D-coordinate locations of the hand of the robotic arm and elaborates run-time a strategy to drive the step-by-step motors of the arm to reach the desired point.

Syntax-driven editor. Realized in SSL language can drive the user to write syntactically correct Pascal programs.

Constructive negation interpreter. Realized in Prolog language, implements the theoretical model "Constructive negation via equational constraint with negation".

Multimedia application. Realized with DIRECTOR 5.0 on Macintosh PowerPC platform for the exhibition *Scultura Lignea*, Lucca, December 95 – June 96. The user can look the technical folders of the sculptures, the pictures taken during the restoration and the location place inside the exhibition rooms.

Abstract diagnosis debugger. Abstract diagnosis consists in comparing a program and its intended meaning and determining the wrong program components. Our debugger do not need to start from symptoms and it systematically derive all the incorrect clauses and uncovered elements, by asking the user questions about the intended meaning of some atomic calls.

This is one of the main issues of my Ph.D. thesis.

Type Verifier. This tool is an evolution of the Abstract diagnosis debugger which can perform verification of Logic Programs w.r.t. type information specifications. The tool is based on sufficient verification conditions obtained by abstract interpretation.

Following the theoretical foundation in some of our papers, the tool is obtained by transforming a static analyzer on a type domain for Logic Programs designed by Codish and Lagoon.

ZipIt. `http://safe-tools.dsic.upv.es/zipit` An Haskell program that computes a compact goal-independent semantics for Term Rewriting Systems as defined in [9]. The tools accepts TRS in two input formats: either TPDB or the equational subset of Maude.

AbsSpec `http://safe-tools.dsic.upv.es/absspec/` A tool to automatically infer specifications from Curry programs. It statically infers from the source code of a Curry program a specification which consists of a set of equations relating (nested) operation calls that have the same behavior. We propose a (white-box) semantic-based inference method which relies on the (fully-abstract condensed) semantics of [43,27,28] for achieving, to some extent, the correctness of the inferred specification.

TRSynth `http://safe-tools.dsic.upv.es/trsynth/` A tool (based on [40,29]) to automatically infer specifications from TRS programs. It statically infers from the source code of a TRS program a specification which consists of a set of equations relating (nested) operation calls that have the same behavior.

6 Teaching and Working Activities

6.1 Thesis Supervision

6.1.1 Supervisor of Master's Theses

A.Y.02-03 "Diagnosi Astratta di Linguaggi Funzionali", Corso di Laurea in Scienze dell'Informazione (Matteo Salsilli).

A.Y.05-06 "Una semantica bottom-up goal-independent per programmi tccp", Corso di Laurea Specialistica in Informatica (Stefano Pramparo).

A.Y.06-07 "Investigazioni su sisitemi di tipi per linguaggi funzionali con ricorsione polimorfa", Corso di Laurea Specialistica in Informatica (Samuel Vrech).

A.Y.06-07 "Filtering approssimato di documenti XML", Corso di Laurea Specialistica in Informatica (Michele Baggi).

A.Y.06-07 "Presentazione ragionata dell'implementazione MCC di Curry", Corso di Laurea Specialistica in Informatica (Marco Girol).

A.Y.07-08 "Diagnosi Astratta di Curry al prim'ordine", Corso di Laurea Specialistica in Informatica (Giovanni Bacci).

A.Y.09-10 "Analisi statica di Sistemi Reattivi Bigrafici tramite Interpretazione Astratta", Corso di Laurea Specialistica in Informatica (Emanuele D'Osualdo).

6.1.2 Supervisor of Batchelor's Theses

A.Y.03-04 (double) "Progettazione ed implementazione di nuovi domini astratti per la verifica di programmi", Corso di Laurea (triennale) in Informatica (Samuel Vrech e Stefano Pramparo).

A.Y.04-05 "Progettazione ed implementazione di un dominio astratto di tipi per linguaggi dichiarativi", Corso di Laurea (triennale) in Informatica (Matteo Dri).

A.Y.04-05 "Uno strumento di Debugging per Full Haskell", Corso di Laurea (triennale) in Informatica (Angelo De Falco).

A.Y.05-06 "Un tool per la conversione semi-automatica di programmi Curry in Haskell e viceversa", Corso di Laurea (triennale) in Informatica (Mauro Jacopo).

A.Y.06-07 "Le costruzioni categoriali dei meccanismi di Haskell", Corso di Laurea (triennale) in Informatica (Emanuele D'Osualdo).

A.Y.06-07 (triple) "Estensione di GHC per supportare Curry", Corso di Laurea (triennale) in Informatica (Dario Meloni, Gianluca Sant e Luca Torella).

A.Y.06-07 "Implementazione distribuita del linguaggio Pascal", Corso di Laurea (triennale) in Informatica (Matteo Cicuttin).

A.Y.07-08 "Semantica Algebrica di CLP(FD)", Corso di Laurea (triennale) in Informatica (Laura Titolo).

6.2 PhD Teaching Activities

2005 Abstract Interpretation and Applications to Program Verification, PhD course in Computer Science, Dipartimento di Matematica e Informatica, *Università di Udine*. [20h]

2006 Abstract Interpretation, PhD course in Computer Science, Dipartimento di Scienze Matematiche e Informatiche "R. Magari", *Università di Siena*. [20h]

6.3 Undergraduate Teaching Activities

A.Y.97-98 Tutorial session of the class Programmazione I (Programming I), Computer Science, Faculty of Sciences, *University of Pisa.* [20h]

A.Y.99-00 Cycle of lectures within the class of Programming I, Computer Science, Faculty of Sciences, *University of Pisa.* [20h]

A.Y.00-01

1. Laboratory of Computer Architecture, Computer Science, Faculty of Sciences, *University of Udine.* [48h]
2. Laboratory of Data Bases, Computer Science, Faculty of Sciences, *University of Udine.* [48h]
3. Introduction to Abstract Interpretation, Computer Science, Faculty of Sciences, *University of Udine.* [8h]

A.Y.01-02

1. Cycle of lectures within the class of Computer Architecture, Computer Science, Faculty of Sciences, *University of Udine.* [48h]
2. Laboratory of Computer Architecture, Computer Science, Faculty of Sciences, *University of Udine.* [48h]
3. Introduction to Abstract Interpretation (within the class of Computer Languages), Computer Science, Faculty of Sciences, *University of Udine.* [8h]
4. Laboratory of Computer Languages (within the class of Computer Languages), Computer Science, Faculty of Sciences, *University of Udine.* [4h]

A.Y.02-03

1. Cycle of lectures within the class of Computer Architecture, Computer Science, Faculty of Sciences, *University of Udine.* [48h]
2. Laboratory of Computer Architecture, Computer Science, Faculty of Sciences, *University of Udine.* [48h]
3. Cycle of lectures and laboratory within the class of Computer Languages I, Computer Science, Faculty of Sciences, *University of Udine.* [8h]
4. Cycle of lectures and laboratory within the class of Computer Languages II, Computer Science, Faculty of Sciences, *University of Udine.* [8h]

A.Y.03-04

1. Laboratory of Computer Architecture (twin courses, A and B), Computer Science, Faculty of Sciences, *University of Udine.* [96h]
2. Esercitazioni . . . Esercitazioni e laboratorio del corso di Linguaggi di Programmazione I, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]
3. Esercitazioni . . . Esercitazioni e laboratorio del corso di Linguaggi di Programmazione II, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]
4. . . . Fondamenti di Informatica (Parte 1 e 2), master I livello "sistemi informativi territoriali", *ENAIP - FVG, Università di Udine, Comune di Tolmezzo, Associazione degli Industriali - Tolmezzo, Agemont* [50h]

A.Y.04-05

1. Formal techniques for Software Engeneering, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
2. Esercitazioni . . . Esercitazioni e laboratorio del corso di Linguaggi di Programmazione I, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]
3. Esercitazioni . . . Esercitazioni e laboratorio del corso di Linguaggi di Programmazione II, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [8h]
4. Operating Systems, Corso di Laurea in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [64h]
5. Programming in the Constraint Logic Paradigm, Scuola Superiore, *Università di Udine.* Reading Course [16h]

A.Y.05-06

1. Formal techniques for Software Engeneering, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
2. Programming Languages I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
3. Programming Languages II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
4. The Functional-Logic Paradigm, Scuola Superiore, *Università di Udine*. Reading Course [16h]

A.Y.06-07

1. Formal techniques for Software Engeneering, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
2. Programming Languages I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
3. Programming Languages II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]

A.Y.07-08

1. Abstract Interpretation and Software Analysis, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
2. Programming Languages I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
3. Programming Languages II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
4. Algorithms and Programming in Declarative Paradigms, Scuola Superiore, *Università di Udine*. Reading Course [16h]
5. Algebraic Semantics of CLP, Scuola Superiore, *Università di Udine*. Reading Course [16h]

A.Y.08-09

1. Abstract Interpretation and Software Analysis, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
2. Programming Languages I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
3. Programming Languages II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
4. Elements of semantics and programming, Scuola Superiore, *Università di Udine*. [16h]
5. Algorithms and Programming in Declarative Paradigms, Scuola Superiore, *Università di Udine*. Reading Course [16h]
6. Insights of course on Elements of semantics and programming, Scuola Superiore, *Università di Udine*. Reading Course [16h]

A.Y.09-10

1. Programming Languages I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]
2. Programming Languages II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine*. [48h]

3. Compilers, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]
4. Algorithms and Programming in Declarative Paradigms, Scuola Superiore, *Università di Udine.* Reading Course [16h]

A.Y.10-11 → now Each year

1. Programming Languages and Compilers I, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [72h]
2. Programming Languages and Compilers II, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [72h]
3. Abstract Interpretation and Automatic Software Verification, Corso di Laurea *Specialistica* in Informatica, Facoltà di Scienze Matematiche, Fisiche e Naturali, *Università di Udine.* [48h]

7 Exam scores

Ph.D. course classes final scores

Teoria della Dimostrazione	**20**/20	(Proof Theory)
Proof Theory and Logic Programming	**20**/20	
Logica Matematica	**20**/20	(Mathematical Logics)
Programmazione Logica	**20**/20	(Logic Programming)
Scuola di Pontignano	**20**/20	(Pontignano's Summer School)
Complessità Computazionale	**20**/20	(Computational Complexity)
Proof Theory of Concurrency	**20**/20	
Verification of Logic and PROLOG programs	**20**/20	
Constraint Programming	**20**/20	
Interpretazione Astratta	**20**/20	(Abstract Interpretation)
Metalogica	**20**/20	(Meta logics)

Scuola Normale Superiore classes final scores

Seminario Fisico Matematico I	**25**/30 (Mathematics and Physics I)
Lettorato di Lingua Russa I	**27**/30 (Russian I)
Lettorato di Lingua Inglese III	**28**/30 (English III)
Seminario Fisico Matematico II	**27**/30 (Mathematics and Physics II)
Lettorato di Lingua Russa II	**24**/30 (Russian II)
Word Problems	**29**/30
Calcolo delle Probabilità	**30**/30 (Probability Theory)
Analisi Superiore	**30**/30 (Advanced Calculus)

Master course classes final scores

Geometria (Geometry)	**30**/30
Teoria ed Applicazioni delle Macchine Calcolatrici (Theory and Applications of Computing Machines)	**30**/30
Teoria degli Algoritmi e della Calcolabilità (Algorithms and Computability Theory)	**27**/30
Algebra	**30**/30 e **lode** (**cum laude**)
Analisi Matematica I (Calculus I)	**29**/30
Sistemi per l'Elaborazione dell'Informazione I (Computer Systems I)	**30**/30 e **lode** (**cum laude**)
Ricerca Operativa e Gestione Aziendale (Operations Research)	**28**/30
Fisica I (Physics I)	**30**/30
Calcolo delle Probabilità e Statistica (Probability Theory and Statistics)	**30**/30
Calcolo Numerico (Numerical Analysis)	**25**/30
Analisi Matematica II (Calculus II)	**27**/30
Metodi per il Trattamento dell'Informazione (Theory of Computation)	**30**/30
Sistemi per l'Elaborazione dell'Informazione II (Computer Systems II)	**29**/30
Linguaggi Speciali di Programmazione (Special Programming Languages)	**30**/30
Fisica II (Physics II)	**30**/30
Linguaggi Formali e Compilatori (Formal Languages and Compilers)	**30**/30 e **lode** (**cum laude**)
Elaborazione dell'Informazione Non Numerica (Artificial Intelligence)	**30**/30 e **lode** (**cum laude**)
Progetto di Sistemi Numerici (Computer Architectures)	**26**/30

Date, May 21, 2013 Signature

(Marco Comini)