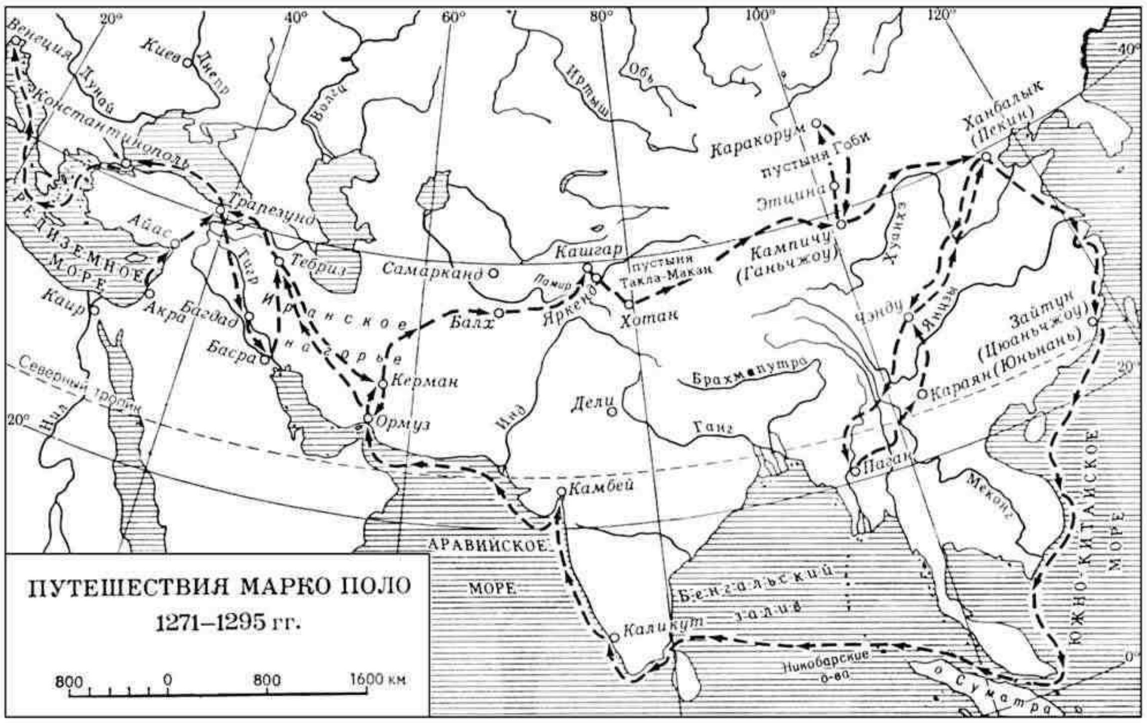


Il Milione *(i.e. 2^6, June 3rd 2008)*

A Journey in the Computational Logic in Italy

Proceedings of the Day Dedicated to Prof. Alberto Martelli

Preface

This volume contains a journey through the work of Italian research groups, sharing an interest to methods and techniques which derive from computational logic. The volume gathers 15 papers that describe the recent experiences of such groups and the future research lines that are considered as particularly promising.

This volume is dedicated to Prof. Alberto Martelli, who is well-known in the community for his studies on heuristic search and for term unification algorithms, for proposals of extensions of logic languages with blocks and modules, for his studies on non-monotonic reasoning techniques, on reasoning about actions and change, and, more recently, on the specification and verification of properties in multi-agent systems and web services.

This collection is not exhaustive, there are many other groups which use computational logic as a research instrument. We are, however, happy to witness the evergreen interest towards computational logic, which proved itself to be a powerful tool with many applications in interesting contexts, ranging from intelligent agent programming to (semantic) web, from the specification and verification of interaction protocols to bioinformatics.

Special thanks to Laura Giordano, Evelina Lamma, Paola Mello, Nicola Olivetti, Viviana Patti, Maria Luisa Sapino e Piero Torasso for their encouragement and support.

June 3rd, 2008

Matteo Baldoni
Cristina Baroglio

Editors

Matteo Baldoni	Università degli Studi di Torino, Italy
Cristina Baroglio	Università degli Studi di Torino, Italy

Authors and Reviewers

Jose Julio Alferes	Universitade Nova de Lisboa, Portugal
Marco Alberti	Università degli Studi di Ferrara, Italy
Matteo Baldoni	Università degli Studi di Torino, Italy
Federico Banti	Universitade Nova de Lisboa, Portugal
Cristina Baroglio	Università degli Studi di Torino, Italy
Piero A. Bonatti	Università di Napoli Federico II, Italy
Annalisa Bossi	Università Ca' Foscari di Venezia, Italy
Antonio Brogi	Università degli Studi di Pisa, Italy
Federico Chesani	Università degli Studi di Bologna, Italy
Nicoletta Cocco	Università Ca' Foscari di Venezia, Italy
Stefania Costantini	Università degli Studi di L'Aquila, Italy
Giorgio Delzanno	Università degli Studi di Genova, Italy
Enrico Denti	Università degli Studi di Bologna, Italy
Agostino Dovier	Università degli Studi di Udine, Italy
Marco Gavanelli	Università degli Studi di Ferrara, Italy
Laura Giordano	Università degli Studi del Piemonte Orientale "Amedeo Avogadro", Italy
Valentina Gliozzi	Università degli Studi di Torino, Italy
Sergio Greco	Università della Calabria, Italy
Evelina Lamma	Università degli Studi di Ferrara, Italy
Nicola Leone	Università della Calabria, Italy
Paolo Mancarella	Università degli Studi di Pisa, Italy
Maurizio Martelli	Università degli Studi di Genova, Italy
Viviana Mascardi	Università degli Studi di Genova, Italy
Paola Mello	Università degli Studi di Bologna, Italy
Marco Montali	Università degli Studi di Bologna, Italy
Nicola Olivetti	Universit Paul Cézanne (Aix-Marseille 3), France
Andrea Omicini	Università degli Studi di Bologna-Cesena, Italy
Luigi Palopoli	Università della Calabria, Italy
Alessio Paolucci	Università degli Studi di L'Aquila, Italy
Viviana Patti	Università degli Studi di Torino, Italy
Alberto Pettorossi	Università degli Studi di Roma "Tor Vergata", Italy
Giulio Piancastelli	Università degli Studi di Bologna-Cesena, Italy
Gian Luca Pozzato	Università degli Studi di Torino, Italy
Maurizio Proietti	Istituto di Analisi dei Sistemi e Informatica "A. Ruberti" (IASI-CNR), Italy

Fabrizio Riguzzi	Università degli Studi di Ferrara, Italy
Gianfranco Rossi	Università degli Studi di Parma, Italy
Pasquale Rullo	Università della Calabria, Italy
Domenico Saccà	Università della Calabria, Italy
Camilla Schwind	CNRS, France
Valerio Senni	Università degli Studi di Roma “Tor Vergata”, Italy
Sergio Storari	Università degli Studi di Ferrara, Italy
Francesca Toni	Imperial College London, United Kingdom
Arianna Tocchio	Università degli Studi di L’Aquila, Italy
Paolo Torroni	Università degli Studi di Bologna, Italy
Panagiota Tsintza	Università degli Studi di L’Aquila, Italy

Table of Contents

Recenti progressi basati su Programmazione Logica e/o a Vincoli sulla soluzione del protein folding

Recent Constraint/Logic Programming based advances in the solution of the Protein Folding Problem

Agostino Dovier

SOMMARIO/*ABSTRACT*

In questo articolo desidero illustrare il contributo del mio gruppo di ricerca alla disciplina Bioinformatica, con particolare riferimento alla risoluzione del problema della predizione di struttura di una proteina usando metodologie di programmazione logica e a vincoli.
In this paper, we summarize the contribution to Bioinformatics of our research group. In particular, we will present our approach to the solution of the protein structure prediction problem based on constraint/logic programming techniques.

Keywords: Logic Programming, Constraint Programming, Bioinformatics

1 Introduction

In the last years we have witnessed the birth and the rapid growth of a new research area whose results have a positive impact on traditional and fundamental disciplines such as biology, chemistry, physics, medicine, agriculture, or industry (briefly denoted globally as "Bio"). This area, known as *Bioinformatics* uses algorithms and methodological techniques developed by Computer Sciences to solve challenging problems in "Bio" areas. Moreover, new emerging problems produce stimuli for Computer Sciences to develop new algorithms and methods. Bioinformatics deals with recognition, analysis, and organization of DNA sequences, with biological systems simulations, with problems of prediction of the spatial conformation of a biological polymer, among others.

We have worked in this field in the last years with the double effort of solving real problems and of spreading known techniques, methods, and languages to "Bio" researchers.

In this spirit, we have been organizers of the workshops WCB (Constraint-Based Methods for Bioinformatics) associated with ICLP in 2005 and 2007, with CP in 2006, and with CPAIOR in 2008 (see, e.g., `http://wcb08.dimi.uniud.it`); we have organized the International Summer Schools BCI (Biology, Communication, and Information) in Dobbiaco and Trieste (see, e.g., `http://bioinf.dimi.uniud.it/bci2006`; and we have been guest editors of a special issue of the journal *Constraints* on these topics [17].

As far as the technical contribution is concerned, we have worked on the Protein Structure Prediction problem using, whenever possible, techniques coming from logic programming and constraint programming. In the rest of this paper we briefly introduce this challenging problem and give an overview of our results.

2 The Protein Structure Prediction Problem

The *Primary structure* of a protein is a linked sequence of aminoacids. There are 20 kinds of aminoacids, identified by a letter. For the scope of this paper, the primary structure of a protein is a string $s_1 \cdots s_n$ with $s_i \in \{A, \ldots, Z\} \setminus \{B, J, O, U, X, Z\}$.

The *Tertiary Structure* (native state) of the protein is a 3D conformation associated to the primary structure. The protein structure prediction problem is the problem of predicting the tertiary structure, given the primary structure.

The Tertiary Structure usually assumes two types of local conformation: α-helices and β-sheets. In Figure 1 we report the primary and the tertiary structure of the protein 2K2P deposited in April 2008. In the top figure all atoms of the amino acids are represented. In the lower figure we report the abstract structure obtained linking the C_α atoms (briefly, a central atom of each aminoacid). With this abstraction the secondary structure elements (three β-sheets and two α-helices) are evident.

Let $\mathcal{D}$ be a set of admissible points for the amino acids. Let c, d two fixed distances. For two points $p, q \in \mathcal{D}$, we say that $\mathsf{next}(p, q)$ if and only if $|p - q| = d$.[1] For two

[1]For real proteins, $d = 3.8\AA$ corresponding to the distance between two consecutive C_α in the sequence

```
A G L S F H V E D M T C G H C A G V I K G
A I E K T V P G A A V H A D P A S R T V V
V G G V S D A A H I A E I I T A A G Y T P E
```

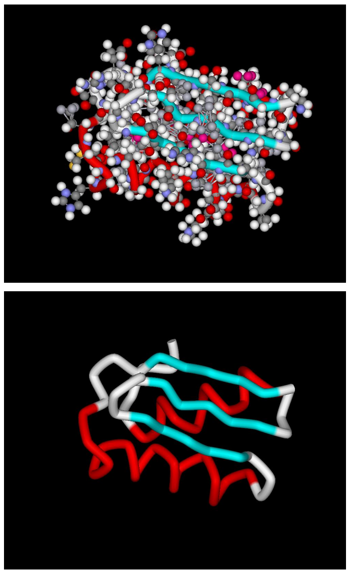


Figure 1: Primary and Tertiary structures (all-atoms and C_α–C_α structure) of Protein 2K2P (amino acids 22–85). Observe the presence of 2 α-helices (in red—dark gray) and 3 β-sheets (in cyan—light gray)

points $p, q \in \mathcal{D}$, we define the Boolean function $\mathsf{contact}$ as follows: $\mathsf{contact}(p, q) = 1$ if and only if $|p - q| \leq c$.

A function $\omega : \{1, \ldots, n\} \longrightarrow \mathcal{D}$ is said a *folding* if

- for $i, j \in \{1, \ldots, n\}$ if $i \neq j$ then $\omega_i \neq \omega_j$
- for $i \in \{1, \ldots, n-1\}$ it holds that $\mathsf{next}(\omega_i, \omega_{i+1})$

Let Pot be a function from pairs of amino acids to integer numbers. The *free energy of a folding* $E(\omega)$ is computed as follows:

$$E(\omega) = \sum_{\substack{1 \leq i < n \\ i+2 \leq j \leq n}} \mathsf{contact}(\omega_i, \omega_j)\mathsf{Pot}(s_i, s_j)$$

The *protein structure prediction problem (PSP)* is the problem of determining the folding(s) ω with minimum energy. The problem contains some symmetries that can be avoided by symmetry breaking search (see e.g. [2]). The simplest way to remove some symmetries is to fix the positions of the first two points (ω_1 and ω_2).

Two main approximations can be made: (1) *space*: the set of admissible points, and (2) *energy*: the details of the Potential function used. It is well-known that lattice-based models are realistic approximations of the set of the admissible points for the C_α atoms of a protein [24]. Lattices are basically 3D graphs with repeated patterns. For instance the *face centered cube (FCC)* lattice is defined as: $\mathcal{D} = \{(x, y, z) \in \mathbb{N}^3 : x + y + z \text{ is even}\}$, $E = \{(p, q) \in \mathcal{D}^2 : |p-q| = \sqrt{2}\}$. Thus, $d = \sqrt{2}, c = 2$.

Three are the main contact energy models used in literature for Pot: the HP model [19], the HPNX model [4], and the 20x20 model [6].

3 Related Work

In the HP model [19], amino acids are split in two families: hydrophobic (H) and polar (P). Two hydrophobic amino acids in contact contribute -1 to the energy. The other contacts are not relevant. The NP-completeness even in the simple spatial model constituted by the $\mathbb{N}^2$ lattice[2] is proved in [9]. In particular, it is proved that the problem: *Given a sequence of P and H, stating the existence of a folding with at least k contacts between H* is NP-complete.

Backofen and Will solved this problem using constraint programming for protein of length 160 and more on the FCC (see [3, 1, 2]). Efficiency is obtained using a clever symmetry breaking and the notion of *core*. Basically, the folding is analyzed layer by layer and the various conformations of each layer that maximize contacts are pre-computed. This kind of approach is unapplicable to a more detailed energy models and with the adding of other structural constraints (e.g., known α-helices and β-sheets). Slightly more complex energy models have been proposed by the same group for the protein structure prediction problem. In [4] they consider an energy model in which amino acids are split into 4 families. Other researchers (e.g. [23]) instead approximated the solution to the same problem using local search and refined meta-heuristics.

Barahona and Krippahl, instead, work on off-lattice space model where space is discretized into small cubes. They also deal with protein docking and develop the tool Chemera, commonly used by biochemists in their research [22, 5].

4 Our Contribution

In all our works, we have used FCC as the space model, and the 20x20 statistical potential contact energy model presented in [6].

CLP(FD) encoding. In [20] we encoded the problem using the library clpfd of SICStus Prolog. Since contact energy is not suited to predict helices and sheets in the FCC lattice, we pre-computed secondary structure elements (α-helices and β-strands) using other well-known tools. The results of these pre-computations were then used as constraints within the main code. In this first encoding the number of admissible angles for secondary structure elements was too limited. We relaxed this restrain in [10] where a more general and precise handling of secondary structure constraints was implemented. However, the exponential growth of the search space w.r.t. protein length made impossible to explore the whole search space

[2] I.e., $\mathcal{D} = \mathbb{N}^2, E = \{(p, q) \in \mathcal{D}^2 : |p - q| = 1\}, c = d = 1$.

even using state-of-the-art constraint solvers for proteins of length greater than 30/40. Therefore, we proposed an ad-hoc labeling search with biologically motivated heuristics and we introduced data structure (potential matrix) that allowed us to reduce calculations during this phase. This approach was then extended by relaxing some constraints and developing other search heuristics [11].

In all these approaches we used a double representation for the tertiary structure: a cartesian one, based on the set of points, and a polar one, based on the torsional angles generated by the protein during the folding. The cartesian representation is useful for defining the notion of self-avoiding walk and the notion of constraint-based energy function. The polar representation simplifies the encoding of secondary structure constraints. However, a lot of extra constraints need to be introduced to manage the conversion between the two representations. This badly scales on large proteins (the constraint solvers used were close to their memory limit for protein of length 60). Thus we decided (in [13]) to abandon the polar representation and to impose secondary structure constraints only using cartesian constraints. This way, we loose the chirality property of helices but the overall definition becomes simpler.

In the same paper we also developed a search heuristics (Bounded Block Fail—BBF). The list of variables is dynamically split into blocks of k variables that will be labeled together. When the variables in the block B_i are instantiated to an apparently admissible solution, the search moves to the successive block B_{i+1}, if any. If the labeling of the block B_{i+1} fails, the search backtracks to the block B_i. Now, there are two options: if the number of times that B_{i+1} has failed is below a certain threshold, then the process continues, by generating one more solution to B_i and re-entering B_{i+1}. Otherwise, the heuristics generates a failure for B_i as well and backtracks to B_{i-1}. The key idea is that small local changes do not change too much the form of a protein. When we tried a sufficient number of close conformations and we fail, we can freely abandon that research branch (with fail we consider either no admissible foldings or admissible foldings with energy greater than the local minimum already found).

Ad-hoc constraint solver. In [12] we developped an ad-hoc constraint solver written in C, named COLA (COnstraint solving on LAttices). In the previous approaches each 3D point was viewed as a triple of FD variables $\langle X, Y, Z\rangle$. In COLA, instead, the lattice point is an elementary element, associated with a 3D domain (a box). We developed and implemented ad-hoc constraint propagation techniques and the BBF heuristics. This approach with a further parallelization was then presented in [16].

Just to give a taste of the evolution of our proposals, we report the running times of the systems on the prediction of some small proteins in Figure 2. Timings are taken from the published papers (the machine used for the leftmost column is roughly 3x slower than the machine for the rightmost). The solutions found with various techniques are not always the same, but (save for the first column related to a too strict encoding) they have comparable energy and form. And, more important, the form is very close to their real tertiary structure. The protein 2K2P of Figure 1 is predicted by COLA 3.1 with BBF in less than one hour.

ID–n	[20]	[10]	[11]	[13]	[16]
1LE3–16	12.5m	5s	2.5s	1.5s	0.5s
1ZDD–34	47m	41s	17.5s	2m	0.1s
2GP8–40	6.5h	9m	10.5h	1.5m	0.5s
1ENH–54	3.5h	13m	24h	55m	49.5s

Figure 2: Running time of the various approaches on some small proteins

Towards generalition and integration. The *ab-initio* approach used by COLA is still computationally infeasible when applied to the prediction of protein structures with more than hundred amino acids. Only the presence of other kind of partial information (e.g., known folds for sub-blocks picked from the protein data bank) can speed-up significantly the search. This is however in line with what done by other prediction tools (like e.g. ROSETTA), where partial information is picked from the protein data-bank from similar structures/substructures and only small subsequences need to be arranged. Thus, we have started a systematic study of what kind of *global constraints* are needed in a solver for lattice models structure predictions. In particular we have studied the definition and the complexity of testing satisfiability and applying propagation for the constraints `alldifferent`, `contiguous`, `self avoiding walk`, `alldistant`, `chain`, and `rigid block` constraint in [14]; we have studied a global constraint that accounts for partial information coming from density maps in [15]. These global constraints will be incorporated in COLA so as to obtain a tool able to profit as much as possible of partial information coming from known proteins and from partial predictions.

We have also studied how to use model checking results for analyzing the folding process [18] and how to model the protein folding problem as a planning problem using a variant of the well-known action description language $\mathcal{B}$ [21]. An alternative approach to the protein folding problem based on Agent-Based simulation is proposed in [7].

5 Conclusions and future work

This work represents a typical use of logic programming paradigm for problem solving. The problem can be encoded easily and solutions (for small inputs) can be computed by built-in mechanisms of (constraint) logic programming. Heuristics and alternative encodings can be easily programmed and tested. When the encoding becomes stable, speed-up can be obtained by less declarative methods. The results obtained are promising for the success of the application of the same approach to other challenging problems of Bioinformatics.

Acknowledgements. The research summarized in this paper would not have been possible without the help of my valuable colleagues and real friends Alessandro Dal Palù, Federico Fogolari, and Enrico Pontelli. I would also like to thank Luca Bortolussi, Elisabetta De Maria, Andrea Formisano, Angelo Montanari, and Carla Piazza for their collaboration. The research has been partially supported by the FIRB RBNE03B8KK, and by PRIN and GNCS projects.

REFERENCES

[1] R. Backofen. The protein structure prediction problem: A constraint optimization approach using a new lower bound. *Constraints* 6:223–255, 2001.

[2] R. Backofen and S. Will. Excluding Symmetries in Constraint-Based Search. *Constraints* 7(3–4):333–349, 2002.

[3] R. Backofen and S. Will. A Constraint-Based Approach to Fast and Exact Structure Prediction in 3-Dimensional Protein Models. *Constraints* 11(1):5–30, 2006.

[4] R. Backofen, S. Will, and E. Bornberg-Bauer. Application of constraint programming techniques for structure prediction of lattice proteins with extended alphabets. *Bioinformatics* 15(3): 234-242, 1999.

[5] P. Barahona and L. Krippahl. Constraint Programming in Structural Bioinformatics. [17]:3–20.

[6] M. Berrera, H. Molinari, and F. Fogolari. Amino acid empirical contact energy definitions for fold recognition in the space of contact maps. *BMC Bioinformatics* 4(8), 2003.

[7] L. Bortolussi, A. Dovier, and F. Fogolari. Agent-based Protein Structure Prediction. *Multiagent and Grid Systems* 3(2):183–197, 2007.

[8] R. Cipriano, A. Palù, and A. Dovier. A hybrid approach mixing local search and constraint programming applied to the protein structure prediction problem. Proc. of WCB08, Paris, 2008.

[9] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the Complexity of Protein Folding, *Journal of Computational Biology*, 5(3):423–466, 1998.

[10] A. Dal Palù, A. Dovier, and F. Fogolari. Protein Folding in CLP(FD) with Empirical Contact Energies. Proc. of CSCLP03:250–265, LNCS 3010, 2004.

[11] A. Dal Palù, A. Dovier, and F. Fogolari. Constraint logic programming approach to protein structure prediction. *BMC Bioinformatics* 5(186), 2004.

[12] A. Dal Palù, A. Dovier, and E. Pontelli. A Constraint Logic Programming Approach to 3D Structure Determination of Large Protein Complexes. Proc. of LPAR05, pp. 48–63, 2005.

[13] A. Dal Palù, A. Dovier, and E. Pontelli. Heuristics, Optimizations, and Parallelism for Protein Structure Prediction in CLP(FD). Proc. of PPDP05, pp. 230–241, ACM, Lisbon 2005.

[14] A. Dal Palù, A. Dovier, and E. Pontelli. Global constraints for Discrete Lattices. Proc. of WCB06, Nantes, pp. 55–68, 2006.

[15] A. Dal Palù, A. Dovier, and E. Pontelli. The density constraint. Proc. of WCB07, Porto, pp. 10–19, 2007.

[16] A. Dal Palù, A. Dovier, and E. Pontelli. A constraint solver for discrete lattices, its parallelization, and application to protein structure prediction. *Software Practice and Experience*, DOI: `10.1002/spe.810`.

[17] A. Dal Palù, A. Dovier, and S. Will (eds.) Special issue on Constraint Based Methods for Bioinformatics. *Constraints* 13(1–2), 2008.

[18] E. De Maria, A. Dovier, A. Montanari, and C. Piazza. Exploiting Model Checking in Constraint-based Approaches to the Protein Folding. Proc. of WCB06, pp.46-54, Nantes, 2006.

[19] K. A. Dill. Dominant forces in protein folding. Biochemistry 29:7133-7155, 1990.

[20] A. Dovier, M. Burato, and F. Fogolari. Using Secondary Structure Information for Protein Folding in CLP(FD). Proc. of WFLP02, ENTCS 76, 2002.

[21] A. Dovier, A. Formisano, and E. Pontelli. Multivalued Action Languages with Constraints in CLP(FD). Proc. of ICLP07, LNCS 4670, pp. 255–270, 2007.

[22] L. Krippahl and P. Barahona. PSICO: Solving Protein Structures with Constraint Programming and Optimisation. *Constraints*, 7:317–331, 2002.

[23] A. Shmygelska and H. H. Hoos. An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. *BMC Bioinformatics* 6(30), 2005.

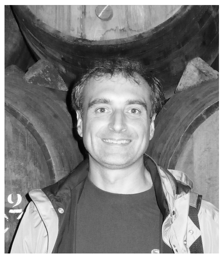
[24] J. Skolnick and A. Kolinski. Reduced models of proteins and their applications. *Polymer*, 45:511–524, 2004.

6 Contacts

Agostino Dovier
Dip. di Matematica e Informatica
Univ. di Udine
Via delle Scienze 206, 33100 Udine (UD)
Tel: +39 0432 558494
E-mail: `dovier@dimi.uniud.it`

7 Biography

Agostino Dovier received his PhD in Computer Science from the University of Pisa in 1996 and he is an Associate Professor of Computer Science at the University of Udine. His current research interests include the development and the application of declarative programming languages with constraints and Bioinformatics. He is member of AI*IA and of the EC of GULP and ALP and he has published over 60 international referred publications. He served as program chair or in the program committee of several conferences and workshops of logic and constraint programming, as guest editor of special issues of international journals, and he has coordinated some research projects in the area of Constraint Logic Programming and Bioinformatics. He is the general chair of ICLP08 (International Conference on Logic Programming).



LA TRASFORMAZIONE DEI PROGRAMMI PER LO SVILUPPO, LA VERIFICA E LA SINTESI DEL SOFTWARE *PROGRAM TRANSFORMATION FOR DEVELOPMENT, VERIFICATION, AND SYNTHESIS OF SOFTWARE*

Alberto Pettorossi, Maurizio Proietti, Valerio Senni

SOMMARIO/*ABSTRACT*

In questo articolo presentiamo brevemente la metodologia di trasformazione dei programmi per lo sviluppo di software corretto ed efficiente. Ci riferiremo, in particolare, al caso della trasformazione e dello sviluppo dei programmi logici con vincoli.

In this paper we briefly describe the use of the program transformation methodology for the development of correct and efficient programs. We will consider, in particular, the case of the transformation and the development of constraint logic programs.

Keywords: Constraint logic programming, model checking, program synthesis, unfold/fold program transformation, software verification.

1 Introduction

The program transformation methodology has been introduced in the case of functional programs by Burstall and Darlington [3] and then it has been adapted to logic programs by Hogger [11] and Tamaki and Sato [26]. The main idea of this methodology is to transform, maybe in several steps and by applying different transformation rules, the given initial program into a final program with the aim of improving its efficiency and preserving its correctness. If the initial program is a *non*-executable specification of an algorithm, while the final program is executable, then program transformation is equivalent to program synthesis. Thus, program transformation can be viewed as a technique both: (i) for *program improvement* and advanced compilation, and (ii) for *program synthesis* and program development.

In recent years program transformation has also been used as a technique for *program verification*. It has been shown, in fact, that via program transformation one can perform *model checking* and, in general, one can prove properties of *infinite* state systems that cannot be analyzed by using standard model checking techniques.

In what follows we will illustrate the three uses of program transformation we have mentioned above, namely (i) program improvement, (ii) program synthesis, and (iii) program verification. In particular, we will consider the case of algorithms and specifications written as *constraint logic programs* [12] and we will focus our attention on the following transformation rules [1, 6, 8, 9, 25, 26]: *definition*, *unfolding*, *folding*, *goal replacement*, and *clause splitting* which will be applied according to some specific strategies. This approach to program transformation is, thus, called the *rules + strategies approach*.

2 Program improvement

Programs are often written in a parametric form so that they can be reused in different contexts, and when a parametric program is reused, one may want to transform it for taking advantage of the new context of use. This transformation, called *program specialization* [10, 13, 16], usually allows a great efficiency improvement. Let us present an example of this transformation by deriving a deterministic, specialized pattern matcher from a given nondeterministic, parametric pattern matcher and a specific pattern. In this example the matching relation on strings of numbers is the relation $le\_m(P,S)$ which holds between a pattern $P=[p_1,\ldots,p_n]$ and a string S iff in S there exists a substring $Q=[q_1,\ldots,q_n]$ such that for $i=1,\ldots,n$, $p_i \leq q_i$. (This example can be generalized by considering any relation which can be expressed via a constraint logic program.)

The following constraint logic program can be taken as the specification of our general pattern matching problem:

1. $le\_m(P,S) \leftarrow ap(B,C,S) \wedge ap(A,Q,B) \wedge le(P,Q)$
2. $ap([\,],\quad s,\quad s) \leftarrow$
3. $ap([X|\quad s],\quad s,[X|\quad s]) \leftarrow ap(\quad s,\quad s,\quad s)$
4. $le([\,],[\,]) \leftarrow$
5. $le([X|\quad s],[Y|\quad s]) \leftarrow X \leq Y \wedge le(Xs,Ys)$

where ap denotes list concatenation. Suppose that we want to use this general program in the case of the pattern $P =$ [1,0,2]. We start off our transformation by introducing the following clause by applying the so-called definition rule:

6. $le\_m_s(S) \leftarrow le\_m([1,0,2], S)$

Clauses 1–6 constitute the initial program from which we begin our program transformation process by applying the transformation rules according to the so-called *Determinization Strategy* [8]. This strategy, which we will not present here, allows a fully automatic derivation of the deterministic, efficient pattern matcher. We unfold clause 6 w.r.t. the atom $le\_m([1,0,2], S)$, that is, we replace the atom $le\_m([1,0,2], S)$ which is an instance of the head of clause 1, by the corresponding instance of the body of clause 1. We get:

7. $le\_m_s(S) \leftarrow ap(B,C,S) \wedge ap(A,Q,B) \wedge le([1,0,2],Q)$

In order to fold clause 7, we introduce the following definition:

8. $ne\quad 1(S) \leftarrow ap(B,C,S) \wedge ap(A,Q,B) \wedge le([1,0,2],Q)$

and then we fold clause 7, that is, we replace (the instance of) the body of a clause 8 which occurs in the body of clause 7 by (the corresponding instance of) the head of clause 8. We get:

9. $le\_m_s(S) \leftarrow ne\quad 1(S)$

Then we unfold clause 8 w.r.t. the atoms *ap* and *le* and we get:

10. $ne\quad 1([X|\quad s]) \leftarrow 1 \leq\quad \wedge ap(Q,C,\quad s) \wedge le([0,2],Q)$
11. $ne\quad 1([X|\quad s]) \leftarrow ap(B,C,\quad s) \wedge ap(A,Q,B) \wedge$ $le([1,0,2],Q)$

Then we apply the clause splitting rule to clause 11, by separating the cases where $1 \leq X$ and $1 > X$. We get:

12. $ne\quad 1([X|\quad s]) \leftarrow 1 \leq X \wedge ap(B,C,\quad s) \wedge ap(A,Q,B) \wedge$ $le([1,0,2],Q)$
13. $ne\quad 1([X|\quad s]) \leftarrow 1 > X \wedge ap(B,C,\quad s) \wedge ap(A,Q,B) \wedge$ $le([1,0,2],Q)$

In order to fold clauses 10 and 12 we introduce the following two clauses defining the predicate $ne\quad 2$:

14. $ne\quad 2(\quad s) \leftarrow ap(Q,C,\quad s) \wedge le([0,2],Q)$
15. $ne\quad 2(\quad s) \leftarrow ap(B,C,\quad s) \wedge ap(A,Q,B) \wedge le([1,0,2],Q)$

Then we fold clauses 10 and 12 by using the two clauses 14 and 15 and we also fold clause 13 by using clause 8. We get the following clauses which define $ne\quad 1$:

16. $ne\quad 1([X|\quad s]) \leftarrow 1 \leq X \wedge ne\quad 2(\quad s)$
17. $ne\quad 1([X|\quad s]) \leftarrow 1 > X \wedge ne\quad 1(\quad s)$

They are mutually exclusive because of the constraints $1 \leq X$ and $1 > X$. Now the program transformation continues in a similar way as above: we introduce the new predicates $ne\quad 3$ through $ne\quad 6$, we derive their defining clauses, and eventually, we get the following specialized, deterministic program:

9. $le\_m_s(S) \leftarrow ne\quad 1(S)$
16. $ne\quad 1([X|\quad s]) \leftarrow 1 \leq X \wedge ne\quad 2(\quad s)$
17. $ne\quad 1([X|\quad s]) \leftarrow 1 > X \wedge ne\quad 1(\quad s)$
18. $ne\quad 2([X|\quad s]) \leftarrow 1 \leq X \wedge ne\quad 3(\quad s)$
19. $ne\quad 2([X|\quad s]) \leftarrow 0 \leq X \wedge 1 > X \wedge ne\quad 4(\quad s)$
20. $ne\quad 2([X|\quad s]) \leftarrow 0 > X \wedge ne\quad 1(\quad s)$
21. $ne\quad 3([X|\quad s]) \leftarrow 2 \leq\quad \wedge ne\quad 5(\quad s)$
22. $ne\quad 3([X|\quad s]) \leftarrow 1 \leq X \wedge 2 > X \wedge ne\quad 3(\quad s)$
23. $ne\quad 3([X|\quad s]) \leftarrow 0 \leq X \wedge 1 > X \wedge ne\quad 4(\quad s)$
24. $ne\quad 3([X|\quad s]) \leftarrow 0 > X \wedge ne\quad 1(\quad s)$
25. $ne\quad 4([X|\quad s]) \leftarrow 2 \leq X \wedge ne\quad 6(\quad s)$
26. $ne\quad 4([X|\quad s]) \leftarrow 1 \leq X \wedge 2 > X \wedge ne\quad 2(\quad s)$
27. $ne\quad 4([X|\quad s]) \leftarrow 1 > X \wedge ne\quad 1(\quad s)$
28. $ne\quad 5([X|\quad s]) \leftarrow$
29. $ne\quad 6([X|\quad s]) \leftarrow$

This final program is deterministic in the sense that at most one clause can be applied during the evaluation of every ground goal. As in the case of the Knuth-Morris-Pratt matcher, the efficiency of this final program is very high because it behaves like a deterministic finite automaton.

3 Program Synthesis

Program synthesis is a technique for deriving programs from formal, possibly *non*-executable, specifications (see, for instance, [11, 24] for the derivation of logic programs from first-order logic specifications). In this section we present an example of use of the program transformation technique for performing program synthesis and deriving a constraint logic program from a first-order formula.

The example we will present is the N-queens example, which has been often considered in the literature for introducing programming techniques such as recursion and backtracking. The N-queens problem can be described as follows. We are required to place $N (\geq 0)$ queens on an $N \times N$ chess board, so that no two queens attack each other, that is, they do not lie on the same row, column, or diagonal. By using the fact that no two queens should lie on the same row, we represent the position of the N queens on the $N \times N$ chess board as a permutation $L = [i_1, \ldots, i_N]$ of the list $[1, \ldots, N]$ such that i_k is the column of the queen on row k.

A specification of the solution L for the N-queens problem is given by the following first-order formula $\varphi(N, L)$:

$nat(N) \wedge nat\_list(L) \wedge length(L,N) \wedge$
$\forall X\, (mem\ er(X,L) \rightarrow in(X,1,N)) \wedge$
$\forall A,B,K,M$
$((1 \leq K \wedge K \leq M \wedge occurs(A,K,L) \wedge occurs(B,M,L))$
$\rightarrow (A \neq B \wedge A - B \neq M - K \wedge B - A \neq M - K))$

where the various predicates which occur in $\varphi(N, L)$ are defined by the following constraint logic program P:

$nat(0) \leftarrow$
$nat(N) \leftarrow N = M + 1 \wedge M \geq 0 \wedge nat(M)$
$nat\_list([\,]) \leftarrow$
$nat\_list([H|T]) \leftarrow nat(H) \wedge nat\_list(T)$

$length([\,],0) \leftarrow$
$length([H|T],N) \leftarrow N = M+1 \wedge M \geq 0 \wedge length(T,M)$
$mem\ \ er(X,[H|T]) \leftarrow X = H$
$mem\ \ er(X,[H|T]) \leftarrow mem\ \ er(X,T)$
$in(X,M,N) \leftarrow X = N \wedge M \leq N$
$in(X,M,N) \leftarrow N = K+1 \wedge M \leq K \wedge in(X,M,K)$
$occurs(X,I,[H|T]) \leftarrow I = 1 \wedge X = H$
$occurs(X,I+1,[H|T]) \leftarrow I \geq 1 \wedge occurs(X,I,T)$

Now, we would like to synthesize a constraint logic program R which computes a predicate $\ ueens(N,L)$ such that the following Property π holds:

$(\pi) \quad M(R) \models \ ueens(N,L)$ iff $M(P) \models \varphi(N,L)$

where by $M(R)$ and $M(P)$ we denote the perfect model of the program R and P, respectively. By applying the technique presented in [9], we start off from the formula $\ ueens(N,L) \leftarrow \varphi(N,L)$. From that formula by applying a variant of the *Lloyd-Topor transformation* [17], we derive this stratified program F:

$\ ueens(N,L) \leftarrow nat(N) \wedge nat\_list(L) \wedge length(L,N) \wedge$
$\qquad \neg aux1(L,N) \wedge \neg aux2(L)$
$aux1(L,N) \leftarrow mem\ \ er(X,L) \wedge \neg in(X,1,N)$
$aux2(L) \leftarrow 1 \leq K \wedge K \leq M \wedge$
$\qquad \neg(A \neq B \wedge A{-}B \neq M{-}K \wedge B{-}A \neq M{-}K) \wedge$
$\qquad occurs(A,K,L) \wedge occurs(B,M,L)$

It can be shown that this variant of the Lloyd-Topor transformation preserves the perfect model semantics and, thus, we have that:

$$M(P \cup F) \models \ ueens(N,L) \text{ iff } M(P) \models \varphi(N,L).$$

The derived program $P \cup F$ is not very satisfactory from a computational point of view, when using SLDNF resolution with the left-to-right selection rule. Indeed, for a query of the form $\ ueens(n,L)$, where n is a nonnegative integer and L is a variable, program $P \cup F$ works by first generating a value l for L and then testing whether or not $length(l,n) \wedge \neg aux1(l,n) \wedge \neg aux2(l)$ holds. This generate-and-test behavior is very inefficient and it may also lead to nontermination. Thus, the process of program synthesis proceeds by applying the definition, unfolding, folding, and goal replacement transformation rules (see [9] for details), with the objective of deriving a more efficient, terminating program. We derive the following definite logic program R:

$\ ueens(N,L) \leftarrow ne\ \ 2(N,L,0)$
$ne\ \ 2(N,[\,],K) \leftarrow N = K$
$ne\ \ 2(N,[H|T],K) \leftarrow \ \ \geq \ \ +1 \wedge ne\ \ 2(N,T,K+1) \wedge$
$\qquad ne\ \ 3(H,T,N,0)$
$ne\ \ 3(A,[\,],N,M) \leftarrow in(A,1,N) \wedge nat(A)$
$ne\ \ 3(A,[B|T],N,M) \leftarrow A \neq B \wedge A - B \neq M+1 \wedge$
$\qquad B - A \neq M+1 \wedge nat(B) \wedge$
$\qquad ne\ \ 3(A,T,N,M+1)$

together with the clauses defining the predicates in and nat.

Since the transformation rules preserve the perfect model semantics, we have that $M(R) \models \ ueens(N,L)$ iff $M(P \cup F) \models \ ueens(N,L)$ and, thus, Property (π) holds. Moreover, it can be shown that R terminates for all queries of the form $\ ueens(n,L)$ and it computes a solution for the N-queens problem in a clever way: each time a queen is placed on the board, program R tests whether or not it attacks every other queen already placed on the board.

4 Program Verification

The proof of program properties is often needed during program development for checking the correctness of software components w.r.t. their specifications. In this section we see the use of program transformation for proving program properties specified either by first-order formulas or by temporal logic formulas.

Proofs performed by using program transformation have strong relationships with proofs by mathematical induction (see [2] for a survey on inductive proofs). In particular, the unfolding rule can be used for decomposing a formula of the form $\varphi(f(X))$, where $f(X)$ is a complex term, into a combination of n formulas of the forms $\varphi_1(X), \ldots, \varphi_n(X)$, and the folding rule can be used for applying inductive hypotheses.

It has been shown that the unfold/fold transformations introduced in [3, 26] can be used for proving several kinds of program properties, such as equivalences of functions defined by recursive equation programs [14], equivalences of predicates defined by logic programs [20], first-order properties of predicates defined by constraint logic programs [21], and temporal properties of concurrent systems [7, 23].

4.1 The unfold/fold proof method

By using a simple example taken from [21], we illustrate a method based on program transformation, called *unfold/fold proof method*, for proving first-order properties of constraint logic programs. Consider the following constraint logic program *Member* which defines the membership relation for lists:

$mem\ \ er(X,[Y|L]) \leftarrow X = Y$
$mem\ \ er(X,[Y|L]) \leftarrow mem\ \ er(X,L)$

Suppose we want to show that every finite list of numbers has an upper bound, i.e., the following formula holds:

$\varphi: \ \forall L\, \exists U\, \forall X\, (mem\ \ er(X,L) \rightarrow X \leq U)$

The unfold/fold proof method works in two steps. In the first step, φ is transformed into a set of clauses by applying a variant of the Lloyd-Topor transformation [17], thereby deriving the following program $Prop_1$:

$prop \leftarrow \neg p$
$p \leftarrow list(L) \wedge \neg q(L)$
$q(L) \leftarrow list(L) \wedge \neg r(L,U)$
$r(L,U) \leftarrow X > U \wedge list(L) \wedge mem\ \ er(X,L)$

The predicate $prop$ is equivalent to φ in the sense that $M(Mem\ \ er) \models \varphi$ iff $M(Mem\ \ er \cup Prop_1) \models prop$.

In the second step, we eliminate the *existential variables* occurring in $Prop_1$ (that is, the variables occurring in the body of a clause and not in its head) by applying the transformation strategy presented in [21]. We derive the following program $Prop_2$ which defines the predicate *prop*:

$prop \leftarrow \neg p$
$p \leftarrow p_1$
$p_1 \leftarrow p_1$

Now, $Prop_2$ is a propositional program and has a *finite* perfect model, which is $\{prop\}$. Since all transformations we have made can be shown to preserve the perfect model, we have that $M(Mem\ \ er) \models \varphi$ iff $M(Prop_2) \models prop$ and, therefore, we have completed the proof of φ because $prop$ belongs to $M(Prop_2)$.

Note that the unfold/fold proof method can be viewed as an extension to constraint logic programs of the *quantifier elimination* method, which has well-known applications in the field of automated theorem proving (see [22] for a brief survey).

4.2 Infinite-state model checking

Now we present a method for verifying temporal properties of infinite state systems based on the transformation of constraint logic programs [7].

As indicated in [4], the behavior of a concurrent system that evolves over time according to a given protocol can be modeled by means of a *state transition system*, that is, (i) a set S of *states*, (ii) an *initial state* $s_0 \in S$, and (iii) a *transition relation* $t \subseteq S \times S$. We assume that t is a *total* relation, that is, for every state $s \in S$ there exists a state $s' \in S$, called *successor state* of s, such that $t(s, s')$ holds. A *computation path* starting from a (possibly not initial) state s_1 is an *infinite* sequence of states $s_1\ s_2 \ldots$ such that, for every $i \geq 1$, there is a transition from s_i to s_{i+1}.

The properties of the evolution over time of a concurrent system are specified by using a temporal logic called *Computation Tree Logic* (or CTL, for short [4]) which specifies the properties of the computation paths. The formulas of CTL are built from a given set of *elementary properties* of the states by using: (i) the connectives: $\neg$ (‘not’) and $\wedge$ (‘and’), (ii) the following quantifiers along a computation path: g (‘for all states on the path’ or ‘globally’),   (‘there exists a state on the path’ or ‘in the future’), x (‘next time’), and u (‘until’), and (iii) the quantifiers over computation paths: a (‘for all paths’) and e (‘there exists a path’).

Very efficient algorithms and tools exist for verifying temporal properties of *finite state systems*, that is, systems where the set S of states is finite [4]. However, many concurrent systems cannot be modeled by finite state systems. Unfortunately, the problem of verifying CTL properties of *infinite* state systems is undecidable, in general, and thus, it cannot be approached by traditional model checking techniques. For this reason various methods based on automated theorem proving have been proposed for enhancing model checking and allowing us to deal with infinite state systems (see [5] for a method based on constraint logic programming). Due to the above mentioned undecidability limitation, all these methods are incomplete.

$$\begin{tikzcd}
\langle t, A2, B1, B2\rangle \arrow[d, "A2 := B2{+}1"'] \\
\langle w, A2, B1, B2\rangle \arrow[d, "A2 < B2 \vee B2 = 0"'] \\
\langle u, A2, B1, B2\rangle \arrow[uu, bend right, "A2 := 0"']
\end{tikzcd}$$

Figure 1: The Bakery protocol: a graphical representation of the transition relation t_A for the agent A.

As an example of use of program transformation for verifying CTL properties of infinite state systems, now we consider the *Bakery* protocol [15] and we verify that it satisfies the *mutual exclusion* and *starvation freedom* properties.

Let us consider two agents A and B which want to access a shared resource in a mutual exclusive way by using the Bakery protocol. The state of agent A is represented by a pair $\langle A1, A2\rangle$, where $A1$ is an element of the set $\{t, w, u\}$ of *control states* (where t, w, and u stand for *think*, *wait*, and *use*, respectively) and $A2$ is a *counter* that takes values from the set of natural numbers. Analogously, the state of agent B is represented by a pair $\langle B1, B2\rangle$. The *state* of the system consisting of the two agents A and B, whose states are $\langle A1, A2\rangle$ and $\langle B1, B2\rangle$, respectively, is represented by the 4-tuple $\langle A1, A2, B1, B2\rangle$. The transition relation t of the two agent system from an old state $\ \ ldS$ to a new state $\ \ e\ \ S$, is defined as follows:

$t(\ \ ldS,\ \ \ e\ \ S) \leftarrow t_A(\ \ ldS,\ \ \ e\ \ S)$
$t(\ \ ldS,\ \ \ e\ \ S) \leftarrow t_B(\ \ ldS,\ \ \ e\ \ S)$

where the transition relation t_A for the agent A is given by the following clauses whose bodies are conjunctions of constraints (see also Figure 1):

$t_A(\langle t, A2, B1, B2\rangle, \langle\ \ , A21, B1, B2\rangle) \leftarrow A21 = B2{+}1$
$t_A(\langle\ \ , A2, B1, B2\rangle, \langle u, A2, B1, B2\rangle) \leftarrow A2 < B2$
$t_A(\langle\ \ , A2, B1, B2\rangle, \langle u, A2, B1, B2\rangle) \leftarrow B2 = 0$
$t_A(\langle u, A2, B1, B2\rangle, \langle t, A21, B1, B2\rangle) \leftarrow A21 = 0$

The following analogous clauses define the transition relation t_B for the agent B:

$t_B(\langle A1, A2, t, B2\rangle, \langle A1, A2,\ \ , B21\rangle) \leftarrow B21 = A2{+}1$
$t_B(\langle A1, A2,\ \ , B2\rangle, \langle A1, A2, u, B2\rangle) \leftarrow B2 < A2$
$t_B(\langle A1, A2,\ \ , B2\rangle, \langle A1, A2, u, B2\rangle) \leftarrow A2 = 0$
$t_B(\langle A1, A2, u, B2\rangle, \langle A1, A2, t, B21\rangle) \leftarrow B21 = 0$

Note that the system has an infinite number of states, because counters may increase in an unbounded way.

The temporal properties of a transition system are specified by defining a predicate $sat(S, P)$ which holds if and only if the temporal formula P is true at state S. For instance, the clauses defining $sat(S, P)$ for the cases where P is: (i) an elementary formula F, (ii) a formula of the form $\neg F$, (iii) a formula of the form $F_1 \wedge F_2$, (iv) a formula of the form $e\ \ (F)$, are the following ones:

$sat(S, F) \leftarrow elem(S, F)$
$sat(S, \neg F) \leftarrow \neg sat(S, F)$
$sat(X, F_1 \wedge F_2) \leftarrow sat(X, F_1) \wedge sat(X, F_2)$
$sat(S, e\ \ (F)) \leftarrow sat(S, F)$
$sat(S, e\ \ (F)) \leftarrow t(S, T) \wedge sat(T, e\ \ (F))$

where $elem(S, F)$ holds iff F is an elementary property which is true at state S. In particular, for the Bakery protocol we have the following clause:

$elem(\langle u, A2, u, B2\rangle, unsa\ e) \leftarrow$

that is, *unsafe* holds at a state where both agents A and B are in the control state u (both agents are accessing the shared resource at the same time).

Note that by *ef* we denote the composition of *e* (there exists a computation path) and *f* (there exists a state on the path) and, indeed, $sat(S, e\ \ (F))$ holds iff there exists a computation path π starting from S and a state s on π such that F is true at s.

The mutual exclusion property holds for the Bakery protocol if there is no computation path starting from the initial state such that at a state on this path the *unsafe* property holds. Thus, the mutual exclusion property holds if $sat(\langle t, 0, t, 0\rangle, \neg e\ \ (unsa\ e))$ belongs to the perfect model $M(P_{mex})$, where: (i) $\langle t, 0, t, 0\rangle$ is the initial state of the system and (ii) P_{mex} is the program consisting of the clauses for the predicates t, t_A, t_B, *sat*, and *elem* defined above.

In order to show that $sat(\langle t, 0, t, 0\rangle, \neg e\ \ (unsa\ e)) \in M(P_{mex})$, we introduce a new predicate mex defined by the following clause:

$(\mu)\quad mex \leftarrow sat(\langle t, 0, t, 0\rangle, \neg e\ \ unsa\ e)$

and we transform the program $P_{mex} \cup \{\mu\}$ into a new program Q which contains a clause of the form $mex \leftarrow$. This transformation is performed by applying the definition, unfolding, and folding rules according to the specialization strategy, that is, a strategy that derives clauses specialized to the computation of predicate mex. From the correctness of the transformation rules we have that $mex \in M(Q)$ iff $mex \in M(P_{mex} \cup \{\mu\})$ and, hence, $sat(\langle t, 0, t, 0\rangle, \neg e\ \ (unsa\ e)) \in M(P_{mex})$, that is, the mutual exclusion property holds.

For the Bakery protocol we may also want to prove the starvation freedom property which ensures that an agent, say A, which requests the shared resource, will eventually get it. This property is expressed by the CTL formula: $ag(\ \ _A \rightarrow a\ \ (u_A))$, which is equivalent to: $\neg e\ \ ((\ \ _A \wedge \neg a\ \ (u_A))$. The clauses defining the elementary properties $\ \ _A$ and u_A are:

$elem(\langle\ \ , A2, B1, B2\rangle, \ \ _A) \leftarrow$
$elem(\langle u, A2, B1, B2\rangle, u_A) \leftarrow$

The clauses defining the predicate $sat(S, P)$ for the case where P is a CTL formula of the form $a\ \ (F)$ are:

$sat(X, a\ \ (F)) \leftarrow sat(X, F)$
$sat(X, a\ \ (F)) \leftarrow ts(X, \ \ s) \wedge sat\_all(\ \ s, a\ \ (F))$
$sat\_all([\,], F) \leftarrow$
$sat\_all([X|\ \ s], F) \leftarrow sat(X, F) \wedge sat\_all(\ \ s, F)$

where $ts(X, \ \ s)$ holds iff $\ \ s$ is a list of all successor states of X. For instance, one of the clauses defining predicate ts in our Bakery example is:

$$ts(\langle t, A2, t, B2\rangle, [\langle\ \ , A21, t, B2\rangle, \langle t, A2, \ \ , B21\rangle]) \leftarrow$$
$$A21 = B2+1 \wedge B21 = A2+1$$

which states that the state $\langle t, A2, t, B2\rangle$ has two possible successor states: $\langle\ \ , A21, t, B2\rangle$ (with $A21 = B2+1$) and $\langle t, A2, \ \ , B21\rangle$ (with $B21 = A2+1$).

Let P_{sf} denote the program obtained by adding to P_{mex} the clauses defining: (i) the elementary properties $\ \ _A$ and u_A, (ii) the atom $sat(X, a\ \ (F))$, (iii) the predicate sat_all, and (iv) the predicate ts. In order to verify the starvation freedom property we introduce the clause:

$(\sigma)\quad s\ \ \leftarrow sat(\langle t, 0, t, 0\rangle, \neg e\ \ (\ \ _A \wedge \neg a\ \ (u_A)))$

and, by applying the definition, unfolding, and folding rules according to the specialization strategy, we transform the program $P_{sf} \cup \{\sigma\}$ into a new program R which contains a clause of the form $s\ \ \leftarrow$.

The derivations needed for verifying the mutual exclusion and the starvation freedom properties were performed in a fully automatic way by using our experimental constraint logic program transformation system MAP [18].

5 Conclusions and Future Directions

We have presented the program transformation methodology and we have demonstrated that it is very effective for: (i) the derivation of correct software modules from their formal specifications, and (ii) the proof of properties of programs. Since program transformation preserves correctness and improves efficiency, it is very useful for constructing software products which are provably correct and whose time and space performance is very high.

In order to make program transformation effective in practice we need to increase the level of automation of the transformation strategies for program improvement, program synthesis, and program verification. Furthermore, these strategies should be incorporated into powerful tools for program development.

An important direction for future research is also the exploration of new areas of application of the transformation methodology. In this paper we have described the use of program transformation for verifying temporal properties of infinite state concurrent systems. Similar techniques could also be devised for verifying other kinds of properties and other classes of systems, such as security properties of distributed systems, safety properties of hybrid systems, and protocol conformance of multiagent systems. A more challenging issue is the fully automatic synthesis of software systems which are guaranteed to satisfy the properties specified by the designer.

6 Acknowledgements

We would like to thank the editors of the Intelligenza Artificiale Magazine for their invitation to write this paper in honor of Prof. Alberto Martelli. We also thank Prof. Martelli for teaching us the 'structural way' of presenting algorithms which we learned from his unification paper [19].

REFERENCES

[1] A. Bossi, N. Cocco, and S. Etalle. Transforming normal programs by replacement. In *Proc. Meta '92*, LNCS 649, 265–279. Springer-Verlag, 1992.

[2] A. Bundy. The automation of proof by mathematical induction. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, 845–911. North Holland, 2001.

[3] R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *J. ACM*, 24(1):44–67, January 1977.

[4] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[5] G. Delzanno and A. Podelski. Constraint-based deductive model checking. *Int. J. Software Tools for Technology Transfer*, 3(3):250–270, 2001.

[6] S. Etalle and M. Gabbrielli. Transformations of CLP modules. *Theo. Comp. Sci.*, 166:101–146, 1996.

[7] F. Fioravanti, A. Pettorossi, and M. Proietti. Verifying CTL properties of infinite state systems by specializing constraint logic programs. In *Proc. VCL'01, Florence (Italy)*, 85–96. Univ. Southampton, UK, 2001.

[8] F. Fioravanti, A. Pettorossi, and M. Proietti. Specialization with clause splitting for deriving deterministic constraint logic programs. In *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics, Hammamet (Tunisia)*, Vol. 1, 188–193. IEEE Computer Society Press, 2002.

[9] F. Fioravanti, A. Pettorossi, and M. Proietti. Transformation rules for locally stratified constraint logic programs. In *Program Development in Computational Logic*, LNCS 3049, 292–340. Springer-Verlag, 2004.

[10] J. P. Gallagher. Tutorial on specialisation of logic programs. In *Proc. PEPM '93, Copenhagen, Denmark*, 88–98. ACM Press, 1993.

[11] C. J. Hogger. Derivation of logic programs. *Journal of the ACM*, 28(2):372–392, 1981.

[12] J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[13] N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.

[14] L. Kott. The McCarthy's induction principle: 'oldy' but 'goody'. *Calcolo*, 19(1):59–69, 1982.

[15] L. Lamport. A new solution of Dijkstra's concurrent programming problem. *Communications ACM*, 17(8):453–455, 1974.

[16] M. Leuschel and M. Bruynooghe. Logic program specialisation through partial deduction: Control issues. *Theory and Practice of Logic Programming*, 2(4&5):461–515, 2002.

[17] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987. Second Edition.

[18] MAP. The MAP transformation system. `http://www.iasi.cnr.it/~proietti/system.html`, 1995–2008.

[19] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM TOPLAS*, 4(12):258–282, 1982.

[20] A. Pettorossi and M. Proietti. Synthesis and transformation of logic programs using unfold/fold proofs. *Journal of Logic Programming*, 41(2&3):197–230, 1999.

[21] A. Pettorossi, M. Proietti, and V. Senni. Proving properties of constraint logic programs by eliminating existential variables. In *Proc. ICLP '06*, LNCS 4079, 179–195. Springer-Verlag, 2006.

[22] M. O. Rabin. Decidable theories. In Jon Barwise, editor, *Handbook of Mathematical Logic*, 595–629. North-Holland, 1977.

[23] A. Roychoudhury, K. Narayan Kumar, C. R. Ramakrishnan, I. V. Ramakrishnan, and S. A. Smolka. Verification of parameterized systems using logic program transformations. In *Proc. TACAS 2000, Berlin, Germany*, LNCS 1785, 172–187. Springer, 2000.

[24] T. Sato and H. Tamaki. Transformational logic program synthesis. In *Proc. Int. Conf. on Fifth Generation Computer Systems*, 195–201. ICOT, 1984.

[25] H. Seki. Unfold/fold transformation of stratified programs. *Theo. Comp. Sci.*, 86:107–139, 1991.

[26] H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In *Proc. ICLP '84*, 127–138, Uppsala, Sweden. Uppsala University, 1984.

7 Contacts

Alberto Pettorossi
Dipartimento di Informatica, Sistemi e Produzione, Università di Roma “Tor Vergata”
Via del Politecnico 1,
00133 Roma, Italy
+39 0672597379
pettorossi@disp.uniroma2.it

Maurizio Proietti (Corresponding Author)
Istituto di Analisi dei Sistemi e Informatica, Consiglio Nazionale delle Ricerche
Viale Manzoni 30,
00185 Roma, Italy
+39 067716426
maurizio.proietti@iasi.cnr.it

Valerio Senni
Dipartimento di Informatica, Sistemi e Produzione, Università di Roma “Tor Vergata”
Via del Politecnico 1,
00133 Roma, Italy
+39 0672597717
senni@disp.uniroma2.it

8 Biography

Alberto Pettorossi is professor of Theoretical Computer Science at the Engineering Faculty of the University of Roma “Tor Vergata”. His research activity has been in the area of rewriting systems and concurrent computation, and it is now concerned with the automatic derivation, transformation, and verification of programs.

Maurizio Proietti received his Laurea degree in Mathematics from the University of Roma “Sapienza”. He is a senior researcher at the Istituto di Analisi dei Sistemi e Informatica “A. Ruberti” of the National Research Council (IASI-CNR) in Roma. His research interests include various aspects of constraint and logic programming, and automatic methods for the transformation, synthesis, and verification of programs.

Valerio Senni is a Ph. D. student in Information Engineering at the the University of Roma “Tor Vergata”. He holds a research contract for the development of automatic methods for proving program correctness.

Photograph. A black and white glossy photograph, passport- sized of each author is also required. It will be printed in the corresponding biography box.

Verso un framework e un linguaggio logico per la programmazione Web
Towards a Logic Language and Framework for Web Programming

Giulio Piancastelli Andrea Omicini Enrico Denti

SOMMARIO/*ABSTRACT*

Nonostante la popolarità del World Wide Web come piattaforma di sviluppo, una adeguata descrizione dei suoi principi architetturali e criteri di progettazione è stata ottenuta solo recentemente, grazie alla introduzione dello stile architetturale REST (Representational State Transfer), che definisce la *risorsa* come la fondamentale astrazione della informazione. Difatti, i linguaggi e gli strumenti correntemente usati per programmare il Web soffrono in genere della mancanza di una corretta comprensione dei suoi vincoli architetturali e progettuali, e di una difformità tra le astrazioni di programmazione che rende difficile sfruttare appieno le potenzialità del Web.
I linguaggi dichiarativi sono particolarmente adatti alla costruzione di sistemi di programmazione rispettosi della architettura e dei principi del Web. Tra le tecnologie di programmazione logica, tuProlog è espressamente progettato per essere uno dei componenti abilitanti di infrastrutture basate su Internet: le sue proprietà ingegneristiche lo rendono peraltro adatto per il Web, dove la programmazione logica permette la modifica del comportamento delle risorse a tempo di esecuzione. Di conseguenza, questo articolo presenta un modello di programmazione logica per risorse Web basato su Prolog e delinea un framework per sviluppare applicazioni Web fondato su quel modello.

Despite the popularity of the World Wide Web as a development platform, a proper description of its architectural principles and design criteria has been achieved only recently, by the introduction of the Representational State Transfer (REST) architectural style which defines the resource *as the key abstraction of information. In fact, languages and tools currently used for Web programming generally suffer from a lack of proper understanding of its architecture and design constraints, and from an abstraction mismatch that makes it hard to exploit the Web potential.*
Declarative languages are well-suited for a programming system aimed at being respectful of the Web architecture and principles. Among logic technologies, tuProlog *has been explicitly designed to be one of the enabling components of Internet-based infrastructures: its engineering properties make it suitable for use on the Web, where logic programming allows modification of resource behaviour at runtime. Accordingly, in this paper we present a Prolog-based logic model for programming Web resources, and outline a framework for developing Web applications grounded on that model.*

Keywords: World Wide Web, REST, Contextual Logic Programming, tuProlog, Prolog.

1 Motivation and Background

Despite the World Wide Web steadily gaining popularity as the platform of choice for the development and fruition of many kinds of Internet-based systems, a proper description of the Web architectural principles and design criteria has been achieved only recently, by the introduction of the Representational State Transfer (REST) architectural style for distributed hypermedia systems [4]. REST defines the *resource* as the key abstraction of information, and prescribes communication and interaction among resources to occur through a *uniform interface* by transferring a *representation* of a resource current state.

Yet, from the early years of procedural CGI scripts to the modern days of object-oriented frameworks, Web application programming has always focussed on different abstractions, such as *page* [5], *controller* [11], and more recently *service* [7], thus suffering from a mismatch that has made it difficult to exploit the full potential of the Web architectural properties. In fact, a page is just the result of a computation involving one or more resources, and deals only with representation issues on the client side. A controller happens to be a programming framework abstraction, sharing almost nothing with the underlying Web platform. Finally, services disregard Web standards such as

URI and HTTP, so they do not get the benefits of the REST architecture in terms of cacheability, connectedness, addressability, uniformity, and interoperability [10].

Declarative programming has never been accepted into the Web mainstream, even though logic languages have shown they could effectively handle both communication and co-ordination in a network-based context [1], and logic technologies have been successfully used to engineer intelligent components at the core of Internet-based infrastructures [2]. However, the REST focus on resource representations as the main driver of interaction, and the corresponding Web computation model, suggest that declarative languages could play a significant role in the construction of resource-oriented applications. The advantage of using elements from logic programming languages such as Prolog lies in the representational foundations of the Web computation model: a declarative representation of resources may be manipulated and, given the procedural interpretation of Prolog clauses, directly executed by an interpreter when a resource is involved in a computation.

Accordingly, we define a resource programming model (called Web Logic Programming [9]), which exploits elements of the logic paradigm and suitable logic technologies (i.e. the tuProlog engine [2]) so as to build a Web application framework aimed at easing rapid prototyping, and allowing the prototype to evolve while supporting Web architectural properties such as scalability or modifiability.

2 Web Logic Programming

Web Logic Programming (WebLP) [9] is a Prolog-based logic model to program resources and their interaction in application systems following the constraints of the World Wide Web architecture. To describe WebLP, we need both to characterise its main data type abstraction and to define its underlying computation model.

2.1 Resources

REST defines a resource as any conceptual target of an hypertext reference. Any information that can be named can be a resource, including virtual (e.g. a document) and non-virtual (e.g. a person) objects. Starting from this abstract definition, the main properties of resources can be easily determined: a name (in the form of an URI); data, representing the resource state; and behaviour, to be used, for instance, to change state or manage the interaction with other resources. The defining elements of resources can be easily mapped onto elements of well-known logic programming languages such as Prolog. For each resource R we specify its name $N(R)$ as the single quoted atom containing the resource URI identifier; data and behaviour can be further recognised as facts and rules, respectively, in a logic theory $T(R)$ containing the knowledge base associated to the resource.

In particular, if adopted resource names are descriptive and have a definite structure varying in predictable ways [10], they feature an interesting property on their own: any path can be interpreted as including a set of resource names. More precisely, we say that resource names such as the following:

```
http://example.com/sales/2004/Q4
```

encompass the names of other resources, and ultimately the name of the resource associated with the domain at the root of the URI:

```
http://example.com/sales/2004
http://example.com/sales
http://example.com
```

This naming structure suggests that each resource does not exist in isolation, but lives in an information *context* composed by the resources associated to the names *encompassed* by the name of that resource.

To account for the possible complexity of Web computations that may involve more information than it is enclosed in a single isolated resource, the context $C(R)$ is introduced as the locus of computation associated with each resource. The context of a resource is defined by the composition of the theories associated with the resources linked to names which are encompassed by that resource name, including the theory associated with the resource itself. Given a resource R with a name $N(R)$ so that:

$$N(R) \subseteq N(R_1) \subseteq \ldots \subseteq N(R_n)$$

the associated context $C(R)$ is generated by composition:

$$C(R) = T(R) \cdot T(R_1) \cdot \ldots \cdot T(R_n)$$

where any theory $T(R_i)$, containing the knowledge base associated to the resource R_i, can be empty – for instance when there is no entity associated to the name $N(R_i)$.

2.2 Computation Model

According to REST, the Web computation model revolves around transactions in the HTTP protocol. Each transaction starts with a *request*, containing the two key elements of Web computations: the *method information*, that indicates how the sender expects the receiver to process the request, and the *scope information*, that indicates on which part of the data set the receiver should apply the method [10]. On the Web, the method information is contained in the HTTP request method (e.g. GET, POST), and the scope information is the URI of the resource to which the request is directed. The result of a Web computation is a *response*, telling whether the request has been successful or not, and optionally containing the representation of the new state of the target resource.

Adopting a logic programming view of the Web computation model, for each HTTP transaction the request can be translated to represent a deduction by retaining the scope information to indicate the target theory, and by mapping the method information onto a proper logic goal. Then, the computation takes place on the server side of the HTTP transaction, in the context associated to the resource target of the request. Finally, the information resulting from goal solution is translated again into a suitable representation and sent back in the HTTP response.

A computation invoked by a goal G on a resource R triggers the deduction of G on the context $C(R)$. The composition of theories forming $C(R)$ is then traversed in a very similar way as units in Contextual Logic Programming (CtxLP) [6]. The goal G is asked in turn to each theory: the goal fails if no solution is found in any theory, or succeeds as soon as it is solved using the knowledge base in a theory $T(R_i)$. Furthermore, when the goal G is substituted by the subgoals of the matching rule in the theory, by default the computation proceeds from $C(R_i)$ rather than being restarted from the original context.

As an example, consider the user `jdoe` in a bookshelf sharing application, where her shelf is represented by the resource S, identified by the URI `http://example.com/jdoe/shelf`. Suppose that, according to a proper naming scheme, the resource B for biology books lives at `/jdoe/shelf/biology`. When a GET request is issued for that resource, a predicate `pick_biology_books/1` is ultimately invoked on B, depending on a `pick_books/3` predicate that is neither defined in B nor in S. The theory chain in $C(B)$ is then traversed backwards up to the `http://example.com` resource, as depicted in Figure 1, where a suitable definition for `pick_books/3` is finally found. Definitions for other predicates invoked by it are then searched starting from the context of the root resource, rather than $C(B)$ where the computation originally started.

The fixed structure of URIs as resource identifiers makes the composition of theories forming a context static, differently from CtxLP, where it was possible to push or pop units from the context stack at runtime. The structure of identifiers and resources in the Web architecture also dictates a unique direction in which the theories associated to resources composing a context can be traversed: from the outermost (associated with the resource on which the computation has been invoked) to the innermost, passing through the theories belonging to each of the composing resources, until the host resource is finally involved.

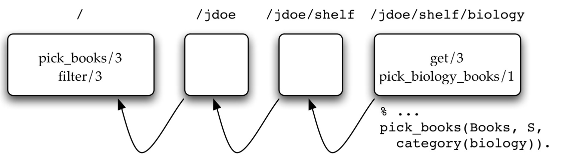


Figure 1: The `/jdoe/shelf/biology` resource responds to a HTTP GET request by eventually invoking the `pick_biology_book/1` predicate, which in turn calls `pick_books/3`. The context is traversed until a proper definition for it is found in the / resource.

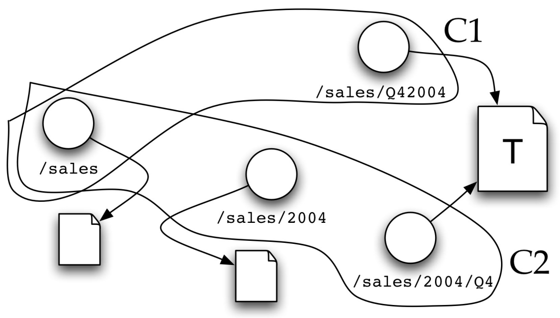


Figure 2: The logic theory of a resource representing sales for the fourth quarter of 2004 can be identified by two different names and therefore live in two different contexts.

2.3 Dynamic Resource Behaviour

The behaviour of a resource can be regarded as dynamic under two independent aspects. First, two or more URIs can be associated to the same resource at any time: that is, the names $N_1(R), \dots, N_m(R)$ may identify the same resource R, thus the same knowledge base contained in the theory $T(R)$ associated to the resource. Each different name $N_i(R)$ also identifies a different context $C_i(R)$ that the same resource R may live within, (see Figure 2); therefore, predicates that are used in $T(R)$, but are not defined there, may behave in different ways based on the definition given by the context where the resource is called.

The second dynamic aspect of a resource comes from the ability to express behavioural rules as first-class abstractions in a logic programming language: on the one hand, well-known logic mechanisms for state manipulation (the `assertz/1` and `retract/1` predicates) can be exploited to change the knowledge base associated to a resource; on the other, the HTTP protocol itself allows changing a resource by means of a PUT request, whose content should be considered as a modified version of the target resource that has to replace (or be merged with) the original version residing on the server.

As an example, imagine a reading wish list in the previous bookshelf application. Usually, when a book is added, the resource representing the wish list could check local libraries for book availability, and possibly borrow it on user's behalf; if no book can be found, the resource could check its availability in online bookstores, reporting its price to the user for future purchase. During the period of time when an online bookstore offers discounts, the wish list resource should react to the insertion of new books so as to check that store first instead of libraries.

The Web application could then be instructed to change the behaviour of wish list resources by issuing HTTP PUT requests that modify the computational representation of those resources. The PUT requests would carry the new rules in the content, so that wish list resources would accordingly modify their knowledge base. The application could programmatically restore the old behaviour at the end of the discount period, by sending another PUT request containing the previous rule set for each wish list resource.

3 tuProlog: Logic Technology for the Web

tuProlog is a minimal Java-based system explicitly designed to integrate configurable and scalable Prolog components into standard Internet applications, and to be used as the core enabling technology for the provision of basic coordination capabilities into complex Internet-based infrastructures [2]. Alongside configurability and scalability, tuProlog has been designed to offer additional engineering properties suitable for distributed systems and architectures: ease of deployment, lightness, and interoperability in accordance with standard protocols (RMI, CORBA, TCP/IP). Those properties are a good match for the architectural properties described by REST, so that tuProlog can be reasonably employed as the core inference engine taking care of resource computations and interactions.

With the aim of sketching a WebLP framework, a minimal Prolog engine such as tuProlog would need to be augmented with a construct very similar to logic contexts, for which various implementation techniques exist, ranging from the least intrusive meta-interpretation to the most effective virtual machine enhancing. With a similar intent, the architecture of tuProlog has been recently re-engineered to feature the malleability property [8], especially important in allowing a light-weight Prolog technology to be extended with similar ease as the Prolog basic execution model can be extended on the pure linguistic side.

The pervasive integration with Java featured by tuProlog [3] is so much important as we consider how much an established platform for Web development Java has become in the latest years. In order to build a WebLP framework, some Java technology which has proven itself effective for some parts of the Web computation model can be exploited, provided that the abstractions underlying the technology are sound within the Web architectural style. As a first example of such technology, the existing Apache Tomcat web server/container can be considered a multi-threaded efficient environment where tuProlog can be integrated, exploiting component life-cycle management and an HTTP uniform interface implementation. JavaServer Pages are a further example, as an extensible technology to produce resource representations to be consumed on the client side of Web applications. Where instead abstractions suffer from mismatch with respect to the REST architectural style, as the case is for Java servlets used as application controllers, they can be dismissed or re-used with a different purpose, for instance as mere HTTP dispatchers.

4 Future Work

The development of tuProlog as a server-side Web technology and of the WebLP framework will be the main focus of our activity in the near future. Afterwards, we also plan to explore possible extensions of the programming model, mostly based on experience in building a variety of applications on the framework.

REFERENCES

[1] E. Denti and A. Omicini. Engineering multi-agent systems in LuCe. In *International Workshop on Multi-Agent Systems in Logic Programming (MAS '99)*, Las Cruces, NM, USA, 30 November 1999.

[2] E. Denti, A. Omicini, and A. Ricci. tuProlog: A light-weight Prolog for Internet applications and infrastructures. In I.V. Ramakrishnan, editor, *Practical Aspects of Declarative Languages*, volume 1990 of *LNCS*, pages 184–198. Springer, 2001.

[3] E. Denti, A. Omicini, and A. Ricci. Multi-paradigm Java-Prolog integration in tuProlog. *Science of Computer Programming*, 57(2):217–250, August 2005.

[4] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[5] R. Lerdorf and K. Tatroe. *Programming PHP*. O'Reilly, April 2002.

[6] L. Monteiro and A. Porto. A Language for Contextual Logic Programming. In *Logic Programming Languages: Constraints, Functions, and Objects*. The MIT Press, 1993.

[7] E. Newcomer and G. Lomow. *Understanding SOA with Web Services*. Addison-Wesley, 2005.

[8] G. Piancastelli, A. Benini, A. Omicini, and A. Ricci. The architecture and design of a malleable object-oriented Prolog engine. In *23th ACM Symposium on Applied Computing (SAC 2008)*, Fortaleza, Ceará, Brazil, 16–20 March 2008.

[9] G. Piancastelli and A. Omicini. A Logic Programming model for Web resources. In *4th International Conference on Web Information Systems and Technologies (WEBIST 2008)*, Funchal, Madeira, Portugal, 4–7 May 2008.

[10] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly, May 2007.

[11] D. Thomas, D. Heinemeier Hansson, L. Breedt, M. Clark, T. Fuchs, and A. Schwarz. *Agile Web Development with Rails*. Pragmatic Bookshelf, 2005.

DALL’UNIFICAZIONE INSIEMISTICA AI VINCOLI SU INSIEMI
FROM SET UNIFICATION TO SET CONSTRAINTS

Gianfranco Rossi

SOMMARIO/*ABSTRACT*

In questo articolo riassumiamo brevemente alcuni dei problemi più interessanti che nascono quando si permette di trattare gli insiemi come oggetti di “prima classe” in un linguaggio logico, dall’unificazione di insiemi ben-fondati e non alla soluzioni di vincoli su insiemi.

In this paper, we briefly summarize some of the most challenging issues that arise when allowing sets to be dealt with as first-class objects in a logic language, ranging from set unification of well-founded and non-well-founded sets to set constraint solving.

Keywords: Set unification, hypersets, set constraints.

1 Introduction

Sets are familiar mathematical objects, and they are often used as a high-level abstraction to represent complex data structures, whenever the order and repetitions of elements are immaterial.

In the last two decades, a number of proposals have emerged where sets are dealt with as primitive objects of a (first-order) logic language. In this context, sets are often represented as first-order terms, called *set terms*, built from symbols of a suitable alphabet, using selected function symbols as set constructors. Furthermore, the language usually provides the typical set-theoretic operations to manipulate set objects.

These short notes summarize some of the most challenging issues arising when manipulating finite sets in a logic language. In Section 2 we briefly review *set unification*, i.e., the key problem of solving equations between set terms. In Section 3 the unification problem is extended to the case of non-well-founded sets. In Section 4 we introduce set constraints as a way to allow set-theoretic operations other than set equality to be taken into account. Finally, in Section 5 we briefly review proposals aiming at making set constraint solving more effective.

2 Set Unification

Intuitively, the set unification problem is the problem of computing (or simply testing the existence of) an assignment of values to the variables occurring in two set terms which makes them denote the same set.

Various forms of set unification have been used in various application areas, such as (see [13]): deductive databases, AI and its various sub-fields (e.g., Automated Deduction and Natural Language Processing), program analysis and security, declarative programming languages with sets.

Set unification can be thought of as an instance of E-unification, i.e., unification modulo an equational theory E, where the identities in E capture the properties of set terms—i.e., the fact that the ordering and repetitions of elements in a set are immaterial.

The equational theory E is strongly related to the representation adopted for set terms. Two main approaches have been presented in the literature: the *union-based representation*, and the *list-like representation*. The union-based representation makes use of the union operator ($\cup$) to construct sets. This representation has been often used when dealing with the problem of set unification on its own, where set unification is dealt with as an ACI unification problem—i.e., unification in presence of operators satisfying the *Associativity*, *Commutativity*, and *Idempotence* properties (e.g., [6]).

The list-like representation builds sets using an *element insertion* constructor (typically denoted by $\{\cdot\,|\,\cdot\}$). With this approach, the finite set $\{t_0, \ldots, t_n\}$ is represented by a sequence of element insertions

$$\{t_0 \,|\, \{\cdots \{t_n \,|\, \emptyset\} \cdots\}\},$$

where $t_0, \ldots, t_n$ are either individuals or sets. While this representation restricts the number of set variables which can occur in each set term to one, on the other hand it allows sets to be viewed and manipulated in a fashion similar to *lists*. As a matter of fact, this representation has been adopted in a number of logic and functional-logic programming languages with sets (e.g., $CLP(\mathcal{SET})$ [10]).

Various authors have investigated the problem of set unification using the list-like representation [3, 12, 23, 8]. In particular, the algorithm presented in [9] considers an equational theory E containing the two identities (Ab) and $(C\ell)$ stating the fundamental properties of the set constructor $\{\cdot\,|\,\cdot\}$:

$$\begin{array}{lccc} (Ab) & \{X\,|\,\{X\,|\,Z\}\} & \approx & \{X\,|\,Z\} \\ (C\ell) & \{X\,|\,\{Y\,|\,Z\}\} & \approx & \{Y\,|\,\{X\,|\,Z\}\}. \end{array}$$

The core of the unification algorithm is very similar in structure to the traditional unification algorithms for standard Herbrand terms (e.g., [20]). The main difference is represented by the reduction of equations between set terms, $\{Y_1\,|\,V_1\} = \{Y_2\,|\,V_2\}$. The algorithm allows also to account for equations of the form $X = \{t_0, \ldots, t_n\,|\,X\}$, with $X \notin vars(t_0, \ldots, t_n)$, which turns out to be satisfiable for any X containing $t_0, \ldots, t_n$ thanks to (Ab) and $(C\ell)$). As an example, given the set unification problem

$$\{X\,|\,S\} = \{1, 2\}$$

(where $\{1, 2\}$ is a syntactic shorthand for $\{1\,|\,\{2\,|\,\emptyset\}\}$) the algorithm non-deterministically computes the following (complete) set of solutions: $X = 1 \wedge S = \{2\}$, $X = 1 \wedge S = \{1, 2\}$, $X = 2 \wedge S = \{1\}$, $X = 2 \wedge S = \{1, 2\}$.

A general survey of the problem of unification in presence of sets, across different set representations and different admissible classes of set terms, can be found in [13].

The computational complexity properties of the set unification have been investigated by Kapur and Narendran [18], who established that these decision problems are NP-complete. Complexity of the set unification operation, however, depends on which forms of set terms (e.g., flat or nested sets, with zero, one, or more set variables) are allowed. The form of set terms in turn is influenced by the set constructors used to build them. Thus, different complexity results can be obtained for different classes of set terms. For instance, while the set equivalence test of ground set terms denoting flat sets, such as $\{a, b, c\}$ and $\{b, c, a\}$, is rather easy, when the decision problem deals with nested set terms involving variables it becomes NP-complete.

Various authors have considered simplified versions of the $(Ab)(C\ell)$ problem obtained by imposing restrictions on the form of the set terms. In particular, various works have been proposed to study the simpler cases of matching[1] and unification of *Bound Simple* set terms, i.e., bound set terms of the form $\{s_1, \ldots, s_n\}$, where each s_i is either a constant or a variable [4, 16].

[1] *Set matching* can be seen as a special case of set unification, where variables are allowed to occur in only one of the two set terms which are compared.

3 Hypersets

Sets considered so far are the so called *hereditarily finite* sets, i.e. sets with a finite number of elements, all of which are themselves hereditarily finite. This definition leaves still a further possibility for infinity. Let us consider the sets x and y that satisfy the equations $x = \{\emptyset, y\}, y = \{x\}$. They are hereditarily finite, but they hide an infinite descending chain $x \ni y \ni x \ni y \ni \cdots$. These sets in which, roughly speaking, membership can form cycles are called *non-well-founded sets* (or *hypersets*). Hypersets are very important in some areas, such as concurrency theory, but hyperset theory has been applied in a number of areas of logic, linguistics, and computer science, as well.

Introducing hypersets as a data structure in a logic programming language requires a unification algorithm that is able to deal with objects denoting hypersets. All set unification algorithms cited in the previous section, however, consider well-founded sets only.

An hyperset unification algorithm is shown in [1]. The key idea underlying this algorithm is that of enlarging the domain of discourse from terms (i.e., finite trees) over the signature Σ to *directed labeled graphs* over Σ, possibly with cycles. This data structure, when involving the interpreted function symbol $\{\cdot\,|\,\cdot\}$ used as the set constructor, can be regarded as a convenient way to denote hypersets. For instance, a solution to the equation $X = \{X\}$ is a cyclic graph which can be interpreted as an hyperset containing itself as its only element. In addition, a notion of *bisimulation* which applies to this kind of graphs is defined and the interpretation domain is taken as the set of directed labeled graphs over Σ modulo the equivalence relation induced by bisimulation.

The algorithm in [1] can be seen as an adaptation of the set unification algorithm of [9]. Many of the changes required to move from set to hyperset unification are the same needed when moving from standard unification to unification over (uninterpreted) *rational trees*, for which a number of efficient algorithms have been proposed in the literature (e.g., [21]). In particular the hyperset unification algorithm in [1] works on Herbrand systems of equations, avoiding full variable substitution and adding simple non-membership constraints to avoid the possibly endless repeated insertions of the same elements into hypersets.

4 Set Constraints

The algorithms cited above focus only on *equality* between set terms. Besides equality, however, other basic set operations, such as membership, inclusion, union, etc., are usually required for dealing with sets in a more general way.

A number of proposals have been put forward in the last fifteen years in which general set-based formulae are considered and procedures to check their consistency are developed. Most of these proposals have emerged in the context of Constraint (Logic) Programming (see, e.g., [15]).

In this context, set-theoretical operations are conveniently dealt with as *(set) constraints*, that is arbitrary conjunctions of positive and negative atomic predicates built using a fixed finite set of predicate symbols denoting set-theoretical operations, whose variables can range over the domain of sets. Systems of (set) constraints are solved as a whole by suitable *(set) constraint solvers*, which are able to reduce the given constraints either to false or to a simplified form from which it is easier to obtain a solution (i.e., a substitution for the variables in the given constraints that make them satisfiable in the selected interpretation). For example,

$$X \in S \wedge T = S \cup R \wedge X \notin T$$

is a set constraint, where R, S, and T are set variables, that the set constraint solver can reduce to false.

Set based formalisms allow a natural formulation of a number of problems, in quite different areas: combinatorial search problems, warehouse location problems, diagnostic related problems (e.g., VLSI circuit verification), program analysis, network design problems (e.g., weight setting). Dealing with such formulations as constraints allow, on the one hand, to solve these problems even if not all sets are (fully) known a priori, and, on the other hand, to compute solutions efficiently, provided constraint reasoning enables the solver to prune the search space.

As an example, the following is a very compact formulation as a set constraint of the well-known map coloring problem for a map of three regions, $R1$, $R2$, $R3$ (where $R1$ borders $R2$ and $R2$ borders $R3$), using two colors, $c1$ and $c2$:

$$\{R1, R2, R3\} = \{c1, c2\} \wedge M = \{\{R1, R2\}, \{R2, R3\}\} \\ \wedge \{c1\} \notin M \wedge \{c2\} \notin M.$$

A complete set constraint solver, i.e., one which is always able to decide if a given constraint is satisfiable or not is presented in [10]. The constraint language is based on constructed sets using the list-like representation and it provides the usual set-theoretic operations as primitive constraints. Sets are allowed to be nested and to contain unknown elements (i.e., uninstantiated logical variables). The constraint solver rewrites any given constraint C into an equi-satisfiable disjunction of constraints in *solved form*—proved to be correct and terminating. In particular the solver uses the set unification algorithm developed in [9] to deal with (set) equalities. A constraint in solved form is guaranteed to be satisfiable in the corresponding structure. Therefore the ability to obtain a solved form guarantees that the original constraint is satisfiable.

This constraint structure has been exploited to obtain a specific instance of the general Constraint Logic Programming scheme, called CLP($\mathcal{SET}$) [10]. A Java implementation of (most of) CLP($\mathcal{SET}$) facilities for set management has been recently developed and made available as part of a Java library, called JSetL [22], intended to support declarative programming in an object-oriented language.

The study of set constraints is strongly related to work in the so-called *Computable Set Theory* area (C.S.T.), a fruitful research stream born in the 1970's at the New York University thanks to the initial ideas and subsequent stimulus of J. T. Schwartz (see [7] for a general survey). Work in C.S.T. has identified increasingly larger classes of computable formulae of suitable sub-theories of the general Zermelo-Fraenkel set-theory for which satisfiability is decidable. Further extensions of these classes are still under investigation at present. Recent related work is described in [19]. However, efforts in this area are mainly concerned with decidability results, rather than computing solutions like it is usually required in constraint programming.

Other classes of aggregates (akin to sets) have also been considered in the literature. In particular, various frameworks have introduced the use of *multisets* where repeated elements are allowed to appear in the collection. An analysis of the problems concerned with the introduction of multisets—as well as sets and lists—is reported in [11, 12].

5 Efficient Set Constraint Solving

The proposals for (general) set constraints cited above do not take into account efficiency adequately to allow them to be effectively applied in many concrete applications. For example the CLP($\mathcal{SET}$) solvers often use a generate & test approach, that non-deterministically assigns values to variables as soon as those values are available. For instance, given the constraint $X \in \{1, 2, 3, 4, 5\} \wedge X \neq 10$, the CLP($\mathcal{SET}$) solver enumerates all possible values of X before asserting that the constraint holds.

A number of proposals have been developed in the last fifteen years that consider more restricted forms of set constraints but equipped with constraint solving techniques that allow them to be processed in a quite more effective way. Works along these lines include [5, 14, 17].

In these proposals constraint variables have a *finite domain* attached to them. In the case of set constraints, the domain is a collection of sets, usually specified as a *set interval* $[l, u]$, where l and u are known sets (typically, of integers). $[l, u]$ represents a lattice of sets induced by the subset partial ordering relation $\subseteq$ having l and u as the greatest lower bound and the least upper bound, respectively. The constraint solver exploits the information that the domain of variables provides to efficiently compute simplified forms of the original constraint or to detect failures. In its simplest form, the solver uses a local propagation algorithm that attempts to enforce consistency on the values in the variable domains by removing values that cannot form part of a solution to the system of constraints. For example, given the set constraint

$$S \in \{1\}..\{1, 2, 3, 4\} \wedge X \subseteq S \wedge Y \subseteq S \\ \wedge \#X = 2 \wedge Z = Y \backslash X$$

where S, X, Y, and Z are set variables and $\#X$ denotes the cardinality of the set X, the constraint solver in [5] is able to infer that the constraint is satisfiable provided $\#Z \leq 2$ holds.

Most of these consistency algorithms are incomplete, so they have to be combined with a backtracking search procedure to produce a complete constraint solver. For example, in the example above, such a procedure allows to enumerate all possible solutions for Z: $Z = \{1\}$, $Z = \{1, 2\}$, $Z = \{1, 3\}$, $Z = \{1, 4\}$.

While these constraint languages turn out to allow more efficient handling of set constraints with respect to the proposals cited in the previous section (e.g., CLP($\mathcal{SET}$)), the latter allows more general form of sets to be dealt with: elements can be of any type, possibly other sets, and possibly unknown (e.g., $\{X, \{a, 1\}\}$). For example the set-based formulation of the map coloring problem shown above can be written—and solved—using CLP($\mathcal{SET}$) but not using the constraint language in [14] and [5].

A current line of research (see [2]) is trying to combine the general set representation and management of proposals like CLP($\mathcal{SET}$), with the efficient constraint solving of "Finite Domain" solvers, in order to have the expressive power of the former while retaining the execution efficiency of the latter.

REFERENCES

[1] D. Aliffi, A. Dovier, and G. Rossi. From Set to Hyperset Unification. *J. of Functional and Logic Programming*, 1999(10):1–48, 1999.

[2] A. Dal Palù, A. Dovier, E. Pontelli, and G. Rossi. Integrating Finite Domain Constraints and CLP with Sets. In D. Miller, ed., *Proc. of 5th ACM-SIGPLAN Int'l Conf. on Principles and Practice of Declarative Programming*, 219–229, ACM Press, 2003.

[3] P. Arenas-Sánchez and A. Dovier. A Minimality Study for Set Unification. *J. of Functional and Logic Programming*, 1997(7):1–49, 1997.

[4] N. Arni, S. Greco, and D. Saccà. Matching of Bounded Set Terms in the Logic Language LDL++. *J. of Logic Programming*, 27(1):73–87, 1996.

[5] F. Azevedo. Cardinal: A Finite Sets Constraint Solver. *Constraints*, 12(37):93–129, 2007.

[6] W. Büttner. Unification in the Data Structure Sets. In J. K. Siekmann, ed., *Proc. of the 8th Int'l Conf. on Automated Deduction*, v. 230, 470–488, Springer-Verlag, 1986.

[7] D. Cantone, E. G. Omodeo, and A. Policriti. *Set Theory for Computing. From Decision Procedures to Declarative Programming with Sets*. Monographs in Computer Science, Springer-Verlag, 2001.

[8] E. Dantsin and A. Voronkov. A Nondeterministic Polynomial-Time Unification Algorithm for Bags, Sets, and Trees. In W. Thomas, ed., *FoSSaCS'99*, *LNCS* 1578, 180–196, Springer-Verlag, 1999.

[9] A. Dovier, E. G. Omodeo, E. Pontelli, and G. Rossi. $\{\log\}$: A Language for Programming in Logic with Finite Sets. *J. of Logic Programming*, 28(1):1–44, 1996.

[10] A. Dovier, C. Piazza, E. Pontelli, and G. Rossi. Sets and Constraint Logic Programming. *ACM TOPLAS*, 22(5):861–931, 2000.

[11] A. Dovier, C. Piazza, and G. Rossi. A uniform approach to constraint-solving for lists, multisets, compact lists, and sets. *ACM Transactions on Computational Logic*, 9(3), 2008.

[12] A. Dovier, A. Policriti, and G. Rossi. A uniform axiomatic view of lists, multisets, and sets, and the relevant unification algorithms. *Fundamenta Informaticae*, 36(2/3):201–234, 1998.

[13] A. Dovier, E. Pontelli, and G. Rossi. Set unification. *Theory and Practice of Logic Programming*, 6:645–701, 2006.

[14] C. Gervet. Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. *Constraints*, 1(3):191–244, 1997.

[15] C. Gervet. Constraints over Structured Domains. In F.Rossi, P. van Beek, and T. Walsh, ed's, *Handbook of Constraint Programming*. Elsevier, 2006.

[16] S. Greco. Optimal Unification of Bound Simple Set Terms. In *Proc. of Conf. on Information and Knowledge Management*, 326–336, ACM Press, 1996.

[17] P. Hawkins, V. Lagoon, and P. J. Stuckey. Solving Set Constraint Satisfaction Problems using ROBDDs. *J. of AI Research*, 24: 109–156, 2005.

[18] D. Kapur and P. Narendran. Complexity of Unification Problems with Associative-Commutative Operators. *J. of Automated Reasoning*, 9:261–288, 1992.

[19] V. Kuncak. Polynomial Constraints for Sets with Cardinality Bounds. *FoSSaCS'99*, *LNCS* 4423, Springer-Verlag, 2007.

[20] A. Martelli and U. Montanari. An Efficient Unification Algorithm. *ACM TOPLAS*, 4:258–282, 1982.

[21] A. Martelli and G. Rossi. Efficient Unification with Infinite Terms in Logic Programming. In *Proc. of FGCS'84: Int'l Conf. on Fifth Generation Computer Systems*, 1984.

[22] G. Rossi, E. Panegai, and E. Poleo. JSetL: A Java Library for Supporting Declarative Programming in Java. *Software-Practice & Experience*, 37:115–149, 2007.

[23] F. Stolzenburg. An Algorithm for General Set Unification and Its Complexity. *J. of Automated Reasoning*, 22(1):45–63, 1999.

6 Contacts

Gianfranco Rossi
Dipartimento di Matematica
Università di Parma
Viale G. P. Usberti 53/A
43100 Parma (Italy)
Phone: +39 (0521) 90.6909
E-mail: gianfranco.rossi@unipr.it

7 Biography

Gianfranco Rossi received a degree in Computer Science from the University of Pisa in 1979. Since November 2001 he is a Full Professor of Computer Science at the University of Parma. His research activity has been mainly devoted to Programming Languages, with special attention to Logic Programming languages. Since December 2006 he is President of the Association of Logic Programming (GULP).

DALI, RASP, Mnemosine: la Logica Computazionale in azione
DALI, RASP, Mnemosine: Computational Logic at Work

Stefania Costantini, Alessio Paolucci, Arianna Tocchio, Panagiota Tsintza

SOMMARIO/*ABSTRACT*

In questo articolo presentiamo le linee di ricerca più recenti del gruppo di Intelligenza Artificiale dell'Universita di L'Aquila. Tali attività riguardano la Logica Computazionale e principalmente gli Agenti Intelligenti logici. Infatti, il gruppo ha sviluppato negli ultimi anni il linguaggio logico orientato agli agenti DALI. Tuttavia, vi sono attività anche in altre aree, come ad esempio il ragionamento non monotono e l'elaborazione del linguaggio naturale. L'attenzione è posta in particolare sulle nuove prospettive di lavoro.

In this paper, we briefly describe recent research directions of the Artificial Intelligence group of the University of L'Aquila, Italy. Research activities concern Computational Logic and mainly Intelligent Logical Agents. In fact, in the last years the group has developed the logical agent-oriented language DALI. However, work is under way also in other areas, like, e.g., Non-Monotonic Reasoning and Natural Language Processing. We particularly emphasize recent and future work directions.

Keywords: Intelligent Agents, Negotiation, Answer Set Programming, Natural Language Processing for the Web.

1 Introduction

Intelligent Agents, computational logic, agents cooperation and negotiation, non-monotonic reasoning, natural language processing and, very recently, biologically inspired models have been the key words of our research activity in the last years[1].

Intelligent agents in computational logic form the "core" of our research. A main achievement of our group has been the definition and development of the DALI language [10, 11, 17, 12], an Active Logic Programming language designed in the line of [15] for executable specification of logical agents. DALI is a prolog-like logic programming language with a prolog-like declarative and procedural semantics. The reactive, proactive and social behavior of DALI agents is triggered by several kinds of events: external, internal, present and past events. The DALI Interpreter has been fully implemented in Sicstus Prolog. DALI agents have been put at work in several real-world applications. An application where the role of DALI agents is particularly relevant has been developed in the context of the CUSPIS European project[2], where DALI agents have been adopted for supporting users during their visit to museums or archeological areas. The system has been practically demonstrated in Villa Adriana (Tivoli, Rome) [8]. However, we have also experimented DALI agents in the context of hybrid architectures and in negotiation scenarios, as summarized in Sections 2 and 3.

Another research line of the group is concerned with Answer Set Programming (ASP for short), which is a form of logic programming based on the answer set semantics [14], where solutions to a given problem are represented in terms of selected models (answer sets) of the corresponding logic program. We have recently proposed RASP, an extension of ASP that permits declarative specification and reasoning on consumption and production of resources, shortly presented in Section 4.

A recent research direction concerns the problem of improving natural language processing by means of parallelizing syntactic and semantic analysis. As a case-study, we have developed a prototype semantic search engine, called *Mnemosine*. This work is outlined in Section 5.

Finally, in the last period we have spent some effort (in cooperation with the biologists of our University) in the study of the human brain as, in view of the growing complexity of computational tasks and their design, many researchers are considering whether interactive systems may be better designed by exploiting computational strategies

[1]The list of publications of the research group can be found at the URL http://www.di.univaq.it/stefcost/pubbls_stefi.htm

[2]CUSPIS, "A Cultural Heritage Space Identification System"(GJU/05/2412/CTR/CUSPIS).

based on the understanding of the human brain. In fact, up to now artificial cognitive systems have been designed without any reference to biology. However, with the predicted increases in computational power and storage capacity over the next decades, it may be important to investigate whether the study of natural cognitive systems can lead to design artificial systems, and in particular agent architectures, with better cognitive capabilities. If so, agent architectures will evolve and agents will become a more effective and useful support for the human activities.

2 Modeling Intelligence in Multi-Layer Cooperative Systems

Nowadays, myriads of heavily networked devices interact with the physical world in multiple ways and at multiple scales. Many of these devices are highly mobile and must adapt to the surrounding environment in a totally unsupervised way.

Biological systems are able to handle many of these challenges with an effectiveness still far beyond current human artifacts: therefore, our long-term goal is to investigate biologically inspired methods on how to engineer intelligent agent systems, so as to exhibit similar high stability and efficiency.

For reconciliating scalability and intelligence, one way is that of creating systems where various degrees of intelligence are distributed over various levels of the architecture. Aim of our research work in this direction has been that of creating a society composed of high-level intelligent agents, aided in their tasks by bunches of elementary agents. The high-level agents are responsible of overall system strategies and plans, to be possibly devised in cooperation. Bunches of elementary agents are supposed to assist each high-level agent in activities where massive parallelism is in order, such as environment exploration, pattern-recognition, classification, action selection and action execution. In fact, as in social insects colonies intelligence emerges from the cooperation activities among individuals, in software environments "intelligence" can be the result of a distribution of roles among different kinds of agents.

We have developed an architecture based on DALI intelligent agents and IBM Aglet mobile agents [1], where the Aglets are Java objects that can move from one host on the Internet to another, and are equipped with communication capabilities: i.e., the Aglets are a very simple kind of agent. The integration of DALI and the Aglets into a colony has been made possible by exploiting those social aspects that are present in both agents platforms. Instead of using standard interfaces, we have implemented a more efficient and flexible communication level not tailored to specific formalisms, that will in perspective allow the integration of other agent platforms into the framework.

As a first experiment of this new kind of architecture, we have tried to model a colony of social insect, though we have widely reinterpreted its structure. We have tried to suitably exploit the features of each platform and to distribute roles among the various entities in order to obtain a kind of "social intelligence" in the artificial colony. Roles that require more "intelligence" are assigned to DALI agents due to their reasoning and learning abilities, while roles requiring communicative and reactive abilities are assigned to Aglets due to their mobility and social nature. In particular, proactivity of DALI agents allowed us to introduce in the artificial colony an entity capable of supervising all the activities which are crucial for the community life like, for example, the planning of some kind of supply or the generation of new individuals useful for the community. Reactivity and mobility of Aglets suggested that their ideal job was that of being the "actuators" of the basic steps of plans devised at the higher level.

We have experimented the architecture in two basic scenarios. In the first one, DALI agents are totally responsible of planning and communicate directly their directions to the actuators. In the second one, DALI agents delegate some planning activity to an intermediate entity that communicates with the actuators. For performing the experiments, we have simulated the activity of bees relative to honey production (where the "honey" the Aglets try to produce may in practical applications correspond to any kind of resource).

The difference between the two scenarios is mainly the degree of interaction, intelligence and sharing among the components. In the first scenario, all managerial and intelligent roles are reserved to the DALI "queen" that organizes the collectivity for producing a certain quantity of honey. According to the queen's indications and using the resources of the society, the colony starts moving, so that every member completes its task with the maximum efficiency. In the second scenario, we have introduced a new role inside the society, i.e., the "courtier". Here, the managerial role previously reserved only to the queen is distributed among the queen and the courtiers. Both scenarios show that the common effort of different entities such as the DALI intelligent agents and the mobile IBM Aglets succeeds in producing coordination so as to reach a common goal.

The number of experiments that we have performed so far does not allow us to establish a general statement. However, the second scenario seem to behave better in the sense that the same quantity of honey is produced in less time. In perspective however, we mean to assess by means of experiments which is the "optimal" distribution of intelligence among levels in relevant classes of applications.

Work is under way in the development of an application of this hybrid architecture in the field of security. As a first step, we have used the Aglets for exploring registries of a computer processor and detecting if an undesired program has been installed. We plan to tackle in the near future other security-related issues.

3 Agents and Negotiation

In the context of Proposal-Based negotiation we have proposed, implemented and experimented an extension of a negotiation approach originally introduced by Marco Cadoli [4].

In the original approach the negotiation areas, representing the admissible values of the negotiation issues, are considered to be convex ones. In particular, the negotiation process is considered as proposal-based with the restriction that at least one of the parts involved in the process is bound to offer only proposal corresponding to vertices of the negotiation area.

This restriction entails that at least one of the negotiation areas has to be polyhedral, while the restriction on the areas to be convex entails that any point of the line connecting two admissible proposals has to be an admissible proposal as well. Possible agreements are represented by the intersection of the two areas. The goal of the approach is to conclude the process, i.e., to find an agreement, by involving the minimum number of interaction between the parties.

The proposed extension [13] is based on relaxing the condition of at least one agent offering vertices of the the negotiation area, which on the one hand may lead to problems if, e.g., the intersection area does not include vertices and on the other hand excludes non-polyhedral areas such as circles. In the extension, proposals can be internal points of the negotiation areas. The points to propose are selected based on the last offer, increased or decreased by a delta margin. A large number of experiments have shown that the proposed extension works properly and that the algorithms performance, in terms of interactions, is reasonable.

Our recent research work on negotiation is related to Argumentation-Based negotiation and its use to cope with contract violations: an agent that has violated an already-signed contract will try to justify this fact by exposing some arguments while the opponent agent will try to undermine their truthfulness and acceptability, by finding attacks against them. As a response, the justifying agent needs in turn to devise a counter-attack. The main theoretical tools that we adopt are logic programming, argumentation and modal logics. Among the objectives of this research are the specification of an appropriate language to support/depict the arguments/justifications used by the agents (we are presently considering dynamic epistemic logic) and the definition of algorithms or mechanisms for performing dialectical disputes among agents.

4 Non-Monotonic Reasoning

Rich literature exists on applications of ASP in many areas, including problem solving, configuration, information integration, security analysis, agent systems, semantic web, and planning (see, e.g., [2] and the references therein). The ASP formulations in these and other fields may take profit from the possibility of performing (at least to some extent) forms of *quantitative* reasoning like those that are possible in, e.g., Linear Logics and Description Logics.

Then, together with Andrea Formisano (University of Perugia) we have recently proposed RASP, an extension of Answer Set Programming that allows for declarative specification and reasoning on consumption and production of resources. Resources are modeled by introducing *amount-atoms*, involving *quantities* that represent the available amount of resources. Processes that use resources are easily described through program rules: in fact, the firing of a RASP-rule can both consume and produce resources. Different solutions correspond to different possible allocations of available resources.

The approach also allows the declarative specification of preferences among alternative uses of available resources. In particular, in realizing the same process (modeled through the firing of a rule), one may prefer to produce a certain product rather than another one and/or to consume certain available resources rather than others. This extension can be particularly useful in planning/configuration applications.

Semantics for RASP programs is provided by combining usual answer set semantics with an interpretation of resource amounts, where different allocation choices correspond to different answer sets.

5 Mnemosine

Menmosine [9] is a prototype semantic search engine based on an extension to the well-known DCGs (Definite Clause Grammars) so as to perform syntactic and semantic analysis to some extent in parallel, and generate semantically-based description of the sentence at hand. The parallelization of syntactic and semantic analysis can help solving some functional deficiencies of classical NLP solutions [3, 16], that often cannot properly cope with ambiguous propositions. In fact, in classical automated language processing methodologies, syntactic and semantic analysis do not have, in many cases, sufficient information neither to determine with certainty the syntactic aspects and details nor to give the correct semantic evaluation. Even probabilistic methods have difficulties.

Thus, for coping with many practical cases NLP systems must at least include ontological reasoning, and thus must become to some extent “intelligent”. Mnemosine relies on an extension to classical DCG’s where a background knowledge base is accessed during the analysis, which is no more divided into separate stages, but can be considered to be “syntactic-semantic”. The input of the analysis is (as usual) a sequence of tokens obtained from lexical analysis. The results of syntactic-semantic analysis consist in (i) establishing the syntactic correctness of the sentence; (ii) creating a formal representation of extracted knowledge; (iii) adding to the knowledge base this representation, as well as the consequences that can be drawn form it. I.e., the objective is to elicit the structure and the meaning of the natural language expression at hand, and to properly exploit it for enlarging or improving the available knowledge.

Mnemosine has been fully implemented and has been applied to a practical case-study, i.e., to the WikiPedia Web pages. We have chosen to use a real data sets from third parties as the choice of data is of primary importance in experiments: in the field of artificial intelligence in fact, many solutions operate properly and efficiently on the data on which they have been developed and tested and then their efficiency collapses dramatically as soon as switched to a real operating environment.

The architecture of Mnemosine has been designed so as to be ready for a timely transformation from a research prototype to an actual industrial product.

6 Conclusions

Throughout the world we are seeing an increased interest in Artificial Intelligence and Computational Logic, despite the relative crisis of Computer Science "per se". Successful results in agents, search and language technology, robotics and web applications are starting the transition to industry. For all this, we have to thank those researchers, like Alberto Martelli, whose important work in computational logic has significantly contributed to the successful development of this field.

REFERENCES

[1] Aglets web site. http://www.trl.ibm.com/aglets/, 2008.

[2] C. Baral. Knowledge representation, reasoning and declarative problem solving. Cambridge University Press, 2003.

[3] P. Blackburn and J. Bos. Representation and Inference for Natural Language. *Natural Language, Center for the Study of Language and Information, StanfordUniversity*, Lecture Notes, 2005.

[4] M. Cadoli, Proposal-Based Negotiation in Convex Regions, *Proceedings of Cooperative Information Agents VII (CIA)*,p.93-108,2003.

[5] S. Costantini, M. S. Del Greco and A. Tocchio. Social bugs communities and Intelligent Agents: an experimental architecture. *Proc. of CILC'08, Ital. Conf. on Computational Logic, Perugia, Italy, July 2008*.

[6] S. Costantini and A. Formisano. Modeling resource production and consumption in answer set programming. Technical Report 04, Dipartimento di Matematica e Informatica, Univertità di Perugia, 2008. Preliminary version in *Proc. of ASP'07, Int.l Worksh. on Answer Set Programming Oporto, Portugal, Sept. 2007*.

[7] S. Costantini and A. Formisano. Modeling preferences on resource consumption and production in answer set programming. *Proc. of CILC'08, Ital. Conf. on Computational Logic, Perugia, Italy, July 2008*.

[8] S. Costantini, L. Mostarda, A. Tocchio and P. Tsintza. DALICA: Agent-Based Ambient Intelligence for Cultural-Heritage Scenarios. *IEEE Intelligent Systems*, vol.23, n.2, pp. 34-41, 2008.

[9] S. Costantini and A. Paolucci. Semantically Augmented DCG Analysis for Next-generation Search Engines. *Proc. of CILC'08, Ital. Conf. on Computational Logic, Perugia, Italy, July 2008*.

[10] S. Costantini and A. Tocchio. A logic programming language for multi-agent systems. *Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf.,JELIA 2002*, LNAI 2424, Springer-Verlag, Berlin,2002.

[11] S. Costantini and A. Tocchio. The DALI logic programming agent-oriented language. *Logics in Artificial Intelligence, Proc. of the 9th European Conference, Jelia 2004*, LNAI 3229,Springer-Verlag, Berlin, 2004.

[12] S. Costantini and A. Tocchio. About declarative semantics of logic-based agent languages. *Declarative Agent Languages and Technologies, Post-Proc. of DALT 2005*, in M. Baldoni and P. Torroni Eds., LNAI 3229, Springer-Verlag, Berlin, 2006.

[13] S. Costantini, A. Tocchio, and P. Tsintza. A Heuristic Approach for P2P Negotiation. *Proc. of the Eighth-Workshop on Computational Logic in Multi-Agent Systems (CLIMA-VIII), September 2007, Porto, Portugal*, http://research.nii.ac.jp/climaVIII.

[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. *Proc. of the 5th Intl. Conference and Symposium on Logic Programming*, pages 1070–1080. The MIT Press, 1988.

[15] A. Kowalski. How to be Artificially Intelligent - the Logical Way *Draft, revised February 2004, Available on line, http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/rak.html*, 2006.

[16] R. Mitkov. The Oxford Handbook of Computational Linguistics. *Natural Language, Center for the Study of Language and Information, StanfordUniversity*, Oxford University Press, 2005.

[17] A. Tocchio. Multi-Agent Systems in Computational Logic. *Ph.D. Thesis, Dipartimento di Informatica, Universitá degli Studi di L'Aquila*, 2005.

7 Contacts

Stefania Costantini (Corresponding Author)
Affiliation: Dept. of Computer Science, Univ. of L'Aquila
Address:Via Vetoio, Loc. Coppito, I-67010 L'Aquila, Italy
Telephone number:+390862433135 E-mail: stefcost@di.univaq.it.

Arianna Tocchio
Affiliation: Dept. of Computer Science, Univ. of L'Aquila
Address:Via Vetoio, Loc. Coppito, I-67010 L'Aquila, Italy
Telephone number:+390862433252
E-mail: tocchio@di.univaq.it.

Panagiota Tsintza
Affiliation: Dept. of Computer Science, Univ. of L'Aquila
Address:Via Vetoio, Loc. Coppito, I-67010 L'Aquila, Italy
Telephone number:+390862433252
E-mail: panagiota.tsintza@di.univaq.it.

8 Biography

Stefania Costantini is a full professor in the University of L'Aquila's Dept. of Computer Science. Her research interests include computational logic and AI, agents and multiagent systems, and answer set programming. She received her master's degree (the highest possible degree in Italy at the time) in computer science from the University of Pisa. She has been Co-hair for the Answer Set Programming 2007 conference.

Alessio Paolucci is a postgraduate student in computer science at the University of L'Aquila. His research interests mainly include Natural Language Processing. He received his Master's degree in Computer Science from the University of L'Aquila.

Arianna Tocchio is a postdoctoral student in computer science at the University of L'Aquila and a research assistant at the university. Her research interests include logical agents and multiagent systems and learning. She received her Master's degree in Computer Science from the University of L'Aquila.

Panagiota Tsintza is a PhD student at the University of L'Aquila. Her research interests are multiagent systems, computational logic agents and automated negotiation. She received her Master's degree in Computer Science from the University of L'Aquila.

Inducing Specification of Interaction Protocols and Business Processes and Proving their Properties
Apprendimento di specifiche di procolli di interazione e processi di business e verica delle loro proprietà

Marco Alberti, Marco Gavanelli, Evelina Lamma, Fabrizio Riguzzi, and Sergio Storari

SOMMARIO/*ABSTRACT*

Questo articolo descrive le nostre recenti attività di ricerca per apprendere (con tecniche di Programmazione Logica Induttiva) specifiche modellate in programmazione logica e per verificare (attraverso una procedura di dimostrazione abduttiva) le proprietà di sistemi così specificati. I sistemi realizzati qui descritti sono stati applicati rispettivamente per l'apprendimento e la verifica di proprietà di protocolli di interazione in sistemi multi-agente, servizi Web, protocolli di screening e processi di business.

In this paper, we overview our recent research activity concerning the induction of Logic Programming specifications, and the proof of their properties via Abductive Logic Programming. Both the inductive and abductive tool here briefly described have been applied to respectively learn and verify (properties of) interaction protocols in multi-agent systems, Web service choreographies, careflows and business processes.

Keywords: Computational logic, Induction, Abduction, Interaction protocols, Careflows, Business processes.

1 Introduction

Thanks to its declarative semantics and its underlying proof theory, Logic Programming, and Computational Logic (CL, for short) in a broader sense, have been proved high-level formal languages for specification and verification. The adoption of logic for computer programming was promoted and improved in the late seventies also in Italy by a clever community. Logic Programming is grounded on a purely declarative representation language, and a theorem-prover or model-generator (like in Answer Set Programming) as the problem-solver. The main task of the problem-solver is the verification that an (existential) query holds in the given specification. Variants of the problem-solver can be also exploited to enrich the representation language and empower the reasoning with new features, such as hypothetical and non-monotonic reasoning, or to prove properties arising from the specification itself. Induction techniques can be also applied, to learn (general and formal) specification from logs and extensional databases or to further abstract specifications.

In this paper, we describe the recent activity carried out at ENDIF, University of Ferrara (also jointly with DEIS, University of Bologna) concerning the induction of CL-based specifications, and the proof of their properties. To this purpose, in the former activity we exploit Inductive Logic Programming techniques (ILP for short), and the DPML algorithm [12] in particular. This algorithm learns a specification expressed in a CL-based language from labeled traces (a database of events recording happened interactions or activities). The target language, named $\mathcal{S}$CIFF, was originally defined for the specification of interaction protocols in the context of the UE IST-2001-32530 Project, and has been later adopted to specify web service choreographies [1], careflows [11] and business processes [5]. A system is specified in the $\mathcal{S}$CIFF language by a knowledge base (a logic program) and a set of $\mathcal{S}$CIFF forward rules, called *integrity constraints*. Each integrity constraint relates occurring events (in the body) with an expected behaviour (typically in the head) in terms of expectations about events. Expectations can be positive (for mandatory events) or negative (forbidden events). Given a $\mathcal{S}$CIFF specification, the compliance of the system to the specifications can be checked on-the-fly through the $\mathcal{S}$CIFF proof-procedure [3], that abduces the expected behaviour and verifies its matching with the actual one.

The adoption of a CL-based language in specifying a system paved also the way to follow a proof-theoretic approach for proving or disproving properties of the given $\mathcal{S}$CIFF specification. To this purpose, we exploit abduction, and in particular an extension of the $\mathcal{S}$CIFF proof-procedure called g-$\mathcal{S}$CIFF [2]. g-$\mathcal{S}$CIFF is an abductive proof-procedure which, starting from a goal, verifies, in a generative manner by abduction, whether there exists a scenario (i.e., a set of generated events) supporting the goal, consistent with the given integrity constraints, and not self-contradictory (e.g., an event does not unify with any forbidden one). In this case, this scenario represents a witness for the goal, and also corresponds to extensions identified by the declarative semantics.

The paper is organized as follows. In Section 2 we briefly introduce the $\mathcal{S}$CIFF language. In Section 3, we show how learning from interpretations can be exploited to learn a $\mathcal{S}$CIFF theory, and also discuss some experimental results. In Section 4, we present g-$\mathcal{S}$CIFF and discuss its application to the learned specification of the previous section. Related work is mentioned throughout the paper. Finally, we conclude in section 5.

This work has been carried out in strict collaboration with the DEIS group. This paper is complementary to [7] contained in this same issue, issue, where they focus on interaction specification and verification in several domains.

2 The $\mathcal{S}$CIFF Language

The $\mathcal{S}$CIFF proof-procedure is an abductive proof procedure, able to reason about dynamically happening events, and to generate corresponding expectations. To represent that an event ev happened (i.e., an atomic activity has been executed) at a certain time T, $\mathcal{S}$CIFF uses the symbol $\mathbf{H}(ev, T)$, where ev is a term and T is a number indicating the time. Hence, an execution trace is modeled as a set of happened events, also called *scenario* or *history* ($\mathbf{HAP}$). For example, we could formalize that bob has performed activity a at time 5 as follows: $\mathbf{H}(a(bob), 5)$. Furthermore, $\mathcal{S}$CIFF introduces the concept of expectation, which plays a key role when defining global interaction protocols, choreographies, and more in general event-driven processes. It is quite natural, in fact, to think of a process in terms of rules of the form: “if A happened, then B is expected to happen”. Positive (resp. negative) expectations are denoted by $\mathbf{E}(ev, T)$ (resp. $\mathbf{EN}(ev, T)$), meaning that ev is expected (resp. expected not) to happen at time T. To satisfy a positive (resp. negative) expectation an execution trace must contain (resp. not contain) a matching happened event.

$\mathcal{S}$CIFF Integrity Constraints (ICs for short) are forward rules of the form $Body \rightarrow Head$:

$$Body \rightarrow Disj_1 \vee \ldots \vee Disj_n \tag{1}$$

where $Body$ is a conjunction of happened events and literals of predicates defined in a $\mathcal{S}$CIFF knowledge base, and $Disj_j$ is a conjunction of expectations (positive and negative) and literals from the knowledge base.

Variables in common to $Body$ and $Head$ are universally quantified ($\forall$) with scope the whole IC. Variables occurring in positive (negative) expectations in $Head(C)$ are existentially (universally) quantified with scope the disjunct where they appear.

An example of an IC is

$$\begin{array}{ll} & \mathbf{H}(a(bob), T) \wedge T < 10 \\ \rightarrow & \mathbf{E}(b(alice), T_1) \wedge T < T_1 \\ \vee & \mathbf{EN}(c(mary), T_1) \wedge T < T_1 \wedge T_1 < T + 10 \end{array} \tag{2}$$

The meaning of the IC (2) is the following: if bob has executed action a at a time $T < 10$, then we either expect $alice$ to execute action b at some time ($\exists T_1$) later than T or we expect that $mary$ does not execute action c at any time ($\forall T_1$) within 9 time units after T.

The interpretation of an IC is the following: if there exists a substitution of variables such that the body is true in an interpretation representing a trace, then one of the disjuncts in the head must be true.

Roughly speaking, $\mathcal{S}$CIFF combines occurred events with the specified rules, to suitably generate the corresponding expectations; then, expectations are verified against the execution trace: a positive expectation must have a corresponding matching event, whereas a negative expectation forbids the presence of a matching event into the trace. If such conditions are not met (i.e., a positive/negative expectation is not/is matched by a corresponding event), then the expectations are violated, and the execution trace is evaluated as non-compliant.

The main and original application of the $\mathcal{S}$CIFF proof-procedure is to verify whether an execution of the process concretely adheres to the specification, i.e., to perform *compliance checking*. $\mathcal{S}$CIFF is seamlessly able to check compliance both at run-time, by dynamically collecting and reasoning upon occurring events, or a-posteriori, by analyzing the log of an observed execution trace.

3 Inducing $\mathcal{S}$CIFF specifications

Since ICs can be seen as an extension of logical clauses, we can apply the techniques developed in the learning from interpretations setting of Inductive Logic Programming [13] to the problem of inducing ICs. In particular, in [12] we modified the Inductive Constraint Logic (ICL) algorithm [9] that takes as input a set of interpretations labeled as positive or negative and returns a clausal theory that is true in as many positive interpretations as possible and false in as many negative interpretations as possible. We called the resulting system DPML [12], for Declarative Process Model Learner.

DPML modifies ICL by replacing the procedure for testing the truth of a clause in an interpretation with a $\mathcal{S}$CIFF-like procedure, by defining a generality order among ICs and, on the basis of this order, by defining a refinement operator. In this way, we can perform search in the space of ICs and evaluate each candidate against the training set.

In DPML the θ-subsumption generality order among clauses is modified in order to take into account the fact that the head is a disjunction of conjunctions. With the new generality relation, we can obtain a generalization D of an IC C by adding a literal to the body, adding a disjunct to the head, removing a literal from a disjunct in the head or adding a literal to a disjunct in the head. This generalization operator is used by DPML to search the space of ICs from specific to general.

The literals to be added are defined by the *language bias*, an intensional definition of the search space. In DPML the language bias is a set of assertions in the form of pairs (BS, HS), where BS is a set that contains the literals that can be added to the body and HS is a set that contains the disjuncts that can be added to the head.

Inducing $\mathcal{S}$CIFF theories is also interesting because it has been shown [6] that other declarative process languages such as DecSerFlow [16] or ConDec [15] can be mapped to $\mathcal{S}$CIFF. Therefore, if we can ensure that the form of the learned ICs corresponds to one of the constraints of these languages, we could learn such constraints by first learning ICs and then translating them into DecSerFlow or ConDec. By providing DPML with a language bias that suitably restricts the search space of ICs, DPML returns a theory with ICs in the desired form, that can be automatically translated into one of the above declarative process languages (see also [11]).

We implemented the whole process of induction plus translation in the DecMiner [11] plug-in of ProM. DecMiner assists the user in all the phases of the learning process, from the definition of the language bias, to the labeling of traces, to the translation of the mined ICs into ConDec constraints.

In particular, the language bias is automatically generated starting from a set of general templates, one for each ConDec constraint, that are then instantiated to generate specific assertions. Since the number of all possible instantiations can be huge, DecMiner asks the user to select a subset of activities A and a subset of ConDec constraints T, and it generates only the instantiations of these constraints with the selected activities.

DPML and DecMiner have been tested on artificial and real datasets. The artificial datasets were randomly generated from three process models, namely the NetBill protocol [8], an electronic auction protocol [4] and a hotel and spa process [11]. The real dataset regards the healthcare process of cervical cancer screening in the Emilia-Romagna Italian region. DPML and DecMiner results were compared with those of the α-algorithm [17] and of the Multi-Phase Miner (MPM) [18] that learn procedural process models.

We now briefly discuss the methodology followed by illustrating the application of DecMiner to the hotel and spa case: the model, inspired by the example presented in [14], describes a simple process of renting rooms and services in a hotel and spa. After registering at the front desk, the client can request one or more rooms, laundry and massage services. Each service, identified by a code, is followed by the registration of the service costs into the client bill. Moreover, if the client chooses a “Shiatzu” massage, the spa presents her/him a special offer. The cost related to the number of nights can be billed before check-out, during check-out or even after check-out.

The $\mathcal{S}$CIFF representation of the hotel model is composed of eight ICs. One of them:

$$\begin{array}{ll} & \mathbf{H}(massage\_service(Type, ma\_id(ID_{ls})), T_{ls}) \\ \rightarrow & \mathbf{E}(bill\_massage\_service(ma\_id(ID_{bls})), T_{bls}) \\ & \wedge ID_{ls} = ID_{bls} \wedge T_{bls} > T_{ls}. \end{array}$$

specifies that a massage service must be followed by the registration of the cost into the client bill.

Five training sets have been generated by randomly building a trace and then classifying it with the ICs of the correct model. The trace is then assigned to the set of positive or negative traces depending on the result of the test. The process is repeated until 2000 positive traces and 2000 negative traces have been generated.

DecMiner, the α-algorithm and MPM were applied to each training set and the learned model was tested on a randomly generated testing set. DecMiner achieved an average accuracy of 99.96%, higher than those of the α-algorithm and MPM.

The sets of ICs returned by DPML/DecMiner can be also used to check (intensional) properties. This can be done by exploiting the g-$\mathcal{S}$CIFF proof-procedure described in the following.

4 Proving properties by g-$\mathcal{S}$CIFF

The $\mathcal{S}$CIFF proof-procedure addresses the important software engineering task of checking compliance during runtime (or a-posteriori using an *event log*), i.e., whether the agents behave in a compliant manner with respect to a given interaction protocol or specification. However, this does not exhaust the possible uses of abductive reasoning: the event literals composing the history can be assumed as well, in order to foresee all the possible evolutions of the system under test. Knowing the specification (in terms of an abductive program), one could (in principle) generate all the histories that the system can support and then study them for common patterns or to formally prove properties of the system.

Of course, explicitly generating all the histories is not feasible, since the number of histories compliant to a protocol are typically infinite for protocols of practical use. However, we can generate compliant histories in intensional way, and then reason upon them: the hypothetical events can contain variables, possibly subject to CLP constraints. In order to generate compliant histories, $\mathcal{S}$CIFF has been improved and extended to a generative version, called g-$\mathcal{S}$CIFF. g-$\mathcal{S}$CIFF considers $\mathbf{H}$ literals as abducibles, and contains a new transition, called *fulfillment*, that fulfils the expectations by abducing matching events:

$$\mathbf{E}(X, T) \rightarrow \mathbf{H}(X, T).$$

g-$\mathcal{S}$CIFF is provably sound: all generated histories fulfil the given specifications.

In the literature, properties are often classified as safety or liveness properties. A *safety* property is a universal property: intuitively, it ensures that nothing bad will ever happen (whenever the protocol/specification is respected). A *liveness* property is, instead, existential: it ensures that something good will eventually happen. A liveness property can be passed to g-$\mathcal{S}$CIFF as a goal containing positive expectations: if the g-$\mathcal{S}$CIFF proof-procedure succeeds in proving the goal, the generated history witnesses that there exists a way to obtain the goal while being conformant to the protocol. A safety property ϕ can be negated (as in model checking), and then passed to g-$\mathcal{S}$CIFF as a goal $\mathcal{G} \equiv \neg\phi$. If the g-$\mathcal{S}$CIFF proof-procedure succeeds in finding a history $\mathbf{HAP}$ (i.e., $\models_{\mathbf{HAP}} \neg\phi$), we have a counterexample: the history $\mathbf{HAP}$ satisfies the protocol and does not enjoy the safety property ϕ.

The g-$\mathcal{S}$CIFF proof-procedure is implemented in SICStus 4, making extensive use of Constraint Handling Rules [10] to implement its transitions. $\mathcal{S}$CIFF and g-$\mathcal{S}$CIFF come in a same package, that can be freely downloaded from the web[1]: the g-$\mathcal{S}$CIFF behaviour is activated by simply setting an option.

The g-$\mathcal{S}$CIFF proof-procedure has been applied to the formal verification of various systems and protocols. g-$\mathcal{S}$CIFF was able to derive the flawedness of the Needham-Schroeder security protocol [2], and the good atomicity property of the NetBill protocol [2]. It is also a basic component of the A^{l}LoWS framework [1], for the proof of interoperability between Web services.

The g-$\mathcal{S}$CIFF proof-procedure operates top-down in a deductive and abductive manner, by manipulating the specification driven by the goal, as usual in Logic Programming, and also generating expectations as $\mathcal{S}$CIFF does and, by *fulfillment* an (intensional) set of events needed to support the goal. This way, g-$\mathcal{S}$CIFF can be used to prove properties of any $\mathcal{S}$CIFF protocol. For example, one may wonder if the protocol allows a massage service not to be followed by a shiatzu package offer. By expressing this combination as a g-$\mathcal{S}$CIFF query, the user can ask g-$\mathcal{S}$CIFF to generate an intensional history that satisfies the query while fulfilling the protocol. In fact, g-$\mathcal{S}$CIFF generates such a history, with the constraint that the massage type must not be shiatzu:

$\mathbf{H}(register\_client\_data, B),$

$\mathbf{H}(massage\_service(type(T), ma\_id(A)), H), T \neq shiatzu$

$\mathbf{H}(bill\_nights, Y),$

$\mathbf{H}(bill\_massage\_service(ma\_id(A)), E),$

$\mathbf{H}(charge, D),$

$\mathbf{H}(complete\_check\_out, F),$

5 Conclusions

We have presented the CL-based language $\mathcal{S}$CIFF for the specifications of complex systems with interacting entities, such as multi-agent systems, business processes or web services. Moreover, we have discussed how techniques from Inductive Logic Programming were applied for inducing $\mathcal{S}$CIFF theories which can be then translated into graphical languages. Finally, the abductive g-$\mathcal{S}$CIFF proof procedure can be used for proving properties of specifications, either learned or provided by the user.

[1] `http://lia.deis.unibo.it/sciff/`

Acknowledgements

We are in debt with the Artificial Intelligence group at DEIS, University of Bologna who shared with us most of the activity here reported.

REFERENCES

[1] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and M. Montali. An abductive framework for a-priori verification of web services. In *PPDP 2006*, pages 39–50. ACM Press, 2006.

[2] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Security protocols verification in abductive logic programming: a case study. In *ESAW 2005*, pages 283–295, 2005.

[3] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Transactions on Computational Logics*, 9(4), 2008. Accepted for publication.

[4] A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *PAAM 1996*, pages 75–90, London, 1996.

[5] F. Chesani, P. Mello, M. Montali, and S. Storari. Testing careflow process execution conformance by translating a graphical language to computational logic. In *Proc. of the* 11^{th} *Conference on Artificial Intelligence in Medicine (AIME 07)*, number 4594 in LNAI, 2007.

[6] F. Chesani, P. Mello, M. Montali, and S. Storari. Towards a DecSerFlow declarative semantics based on computational logic. Technical Report DEIS-LIA-07-002, University of Bologna, 2007.

[7] F. Chesani, P. Mello, M. Montali, and P. Torroni. Modeling and verification of business processes and choreographies in ALP. *Intelligenza Artificiale*. In this issue.

[8] B. Cox, J.C. Tygar, and M. Sirbu. NetBill security and transaction protocol. In 1^{st} *USENIX Workshop on Electronic Commerce*, 1995.

[9] L. De Raedt and W. Van Laer. Inductive constraint logic. In *ALT 1995*, volume 997 of *LNAI*, 1995.

[10] T. Frühwirth. Theory and practice of constraint handling rules. *J. of Logic Prog.*, 37(1-3):95–138, 1998.

[11] E. Lamma, P. Mello, M. Montali, F. Riguzzi, and S. Storari. Inducing declarative logic-based models from labeled traces. In *BPM 2007*, volume 4714 of *LNCS*, 2007.

[12] E. Lamma, P. Mello, F. Riguzzi, and S. Storari. Applying inductive logic programming to process mining. In *ILP 2007*, volume 4894 of *LNAI*, 2007.

[13] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *J. Logic Prog.*, 19/20:629–679, 1994.

[14] M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. Declare: Full support for loosely-structured processes. In *EDOC 2007*, pages 287–300. IEEE Computer Society, 2007.

[15] W.M.P. van der Aalst and M. Pesic. A declarative approach for flexible business processes management. In *BPM 2006*, volume 4103 of *LNCS*, 2006.

[16] W.M.P. van der Aalst and M. Pesic. DecSerFlow: Towards a truly declarative service flow language. In *WS-FM 2006*, volume 4184 of *LNCS*, 2006.

[17] W.M.P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.

[18] B. F. van Dongen and W. M. P. van der Aalst. Multi-phase process mining: Building instance graphs. In *ER 2004*, volume 3288 of *LNCS*, pages 362–376, 2004.

Contacts

Marco Alberti, marco.alberti@unife.it
Marco Gavanelli, marco.gavanelli@unife.it
Evelina Lamma, evelina.lamma@unife.it
Fabrizio Riguzzi, fabrizio.riguzzi@unife.it
Sergio Storari, sergio.storari@unife.it

tutti affiliati al

Dipartimento di Ingegneria,
Università di Ferrara
Via Saragat, 1
44100 Ferrara

Computational Logic in Genova
Logica Computazionale a Genova

Viviana Mascardi, Giorgio Delzanno, Maurizio Martelli
DISI, Università degli Studi di Genova,
Via Dodecaneso 35, 16146, Genova, Italy
E-mail: {viviana.mascardi,giorgio.delzanno,
maurizio.martelli}@unige.it

SOMMARIO/*ABSTRACT*

La Logica Computazionale gioca un ruolo molto rilevante nella ingegnerizzazione di sistemi complessi: può essere usata per specificare sistemi al livello di astrazione più opportuno, la specifica può essere eseguita fornendo gratuitamente un prototipo funzionante e, grazie alla sua semantica ben fondata, può essere usata per verificare formalmente proprietà di programmi e sistemi, cosa fondamentale nello sviluppo di applicazioni critiche dal punto di vista della sicurezza.

Nell'ultimo decennio, il Gruppo di Programmazione Logica del Dipartimento di Informatica e Scienze dell'Informazione (DISI) dell'Università degli Studi di Genova ha applicato la Logica Computazionale per modellare, prototipare e verificare sistemi complessi. Le tre linee di ricerca hanno ampie aree di sovrapposizione: i sistemi complessi che prendiamo in considerazione sono spesso sistemi multiagente per i quali proponiamo linguaggi di modellazione, ambienti di prototipazione e tecniche di verifica. Inoltre usiamo la logica temporale sia per modellare agenti BDI cooperativi, sia per verificare processi a stati infiniti.

In questo articolo descriviamo le attività condotte recentemente in ciascuna direzione di ricerca.

Computational Logic plays a very relevant role in engineering complex systems: it can be used to specify systems at the right level of abstraction, the specifications can be executed, thus providing a working prototype for free, and thanks to its well-founded semantics it can be used to formally verify properties of programs, which is fundamental when safety critical applications are developed.

In the last decade, the Logic Programming Group at the Department of Computer and Information Science (DISI) of Genova University has been applying Computational Logic for modelling, prototyping, and verifying complex systems. These three research lines are largely overlapping: the complex systems we take under consideration are often multiagent systems, for which we propose modelling languages as well as prototyping environments and verification techniques. Also, we use temporal logic both for modelling cooperative BDI agents and for verifying infinite-state processes.

In this paper, we describe the activities that we carried out in the recent years in each research line.

Keywords: Computational Logic, Intelligent Agents, Rapid Prototyping, Verification of Protocols.

Logic Languages for Modelling Rational Agents

Many logics for modelling beliefs, desires and intentions of agents, such as Rao and Georgeff's BDI logic [36, 34, 35] and Wooldridge's $\mathcal{LORA}$ [40], are based on temporal logics like CTL/CTL* (Computational Tree Logic, [24, 17]) where the structure of time is branching in the future and linear in the past. In 2005 we started to explore the advantages of substituting ATL* (Alternating-Time Temporal Logic [1]) to CTL* in Rao and Georgeff's logic. This activity, resulted into the formalization of BDI^{ATL} [33], was born from our effort to find a BDI logic suitable for modelling the behaviour of agents structured according to the CooBDI architecture [2].

A CooBDI agent, whose behavioral specification was given using Prolog, is characterised by a built-in mechanism for retrieving plans from cooperative agents, for example when no local plans suitable for achieving a certain desire are available. In particular, the cooperation strategy of an agent includes the set of agents with which is expected to cooperate (its partner agents, or its "friends"). BDI^{ATL} allows us to express new commitment strategies that are more realistic than those proposed by Rao and Georgeff (and that could not be defined in their logic), since they take collaboration among agents into account. In particular, we can express three variants of Rao and Georgeff's "open minded" commitment: "independent open minded", "optimistic open minded", and "pessimistic open minded". In these commitment strategies we exploit the new feature that ATL* adds to CTL*, namely *cooperation modalities*, to express the way of thinking of CooBDI agents.

Other logic-based languages conceived for specifying BDI-style and, more in general, rational agents, are CongoLog [27], AGENT-0 [38], Concurrent METATEM [25], $\mathcal{E}_{hhf}$ [20], the IMPACT language [23], and "Dynamics in Logic" [10]. In 2004, we published a survey of these six languages [32], chosen because of the availability, for each of them, of a working interpreter or an automatic mechanism for animating specifications. In our survey we described the logic foundations of each language and we gave an example of use. A comparison along twelve dimensions (purpose of use, language support to time, sensing, concurrency, nondeterminism, etc.) was also provided.

Computational Logic for MAS Prototyping

It is well known that computational logic and logic programming in particular are very suitable to implement sophisticated, self-aware agents able to reason about themselves and the other agents in a multiagent system (MAS). DCaseLP (*Distributed Complex Applications Specification Environment based on Logic Programming* [31]) is an environment for rapid prototyping of MASs developed by the Logic Programming Group at DISI. DCaseLP was initially born as a logic-based framework, as the acronym itself suggests, and then evolved into a multi-language prototyping environment that integrates both imperative (object-oriented) and declarative (rule-based and logic-based) languages, as well as graphical ones. The languages and tools that DCaseLP integrates are UML and an XML-based language for the analysis and design stages, Java, JESS [26] and tuProlog [22] for the implementation stage, and JADE [12] for the execution stage. Software libraries for integrating JESS and tuProlog agents into the JADE platform and for translating UML class diagrams into JESS and tuProlog code are also provided[1]. The methodological integration of DCaseLP with the "Dynamics in Logic" agent programming language is described in [6].

All the applications that we developed with DCaseLP in collaboration with Italian industries, exploit tuProlog for implementing the MAS.

The most recent application, described in [30], is a MAS that monitors processes running in a railway signalling plant, detects functioning anomalies, provides diagnoses for explaining them, and early notifies problems to the Command and Control System Assistance. This work is part of an ongoing project that involves DISI and Ansaldo Segnalamento Ferroviario, the Italian leader in design and construction of signalling and automation systems for railway lines.

[1]The source code of DCaseLP libraries together with manuals and tutorials is available from `http://www.disi.unige.it/person/MascardiV/Software/DCaseLP.html`.

The work described in [37] deals with an electronic implementation of different auction mechanisms. There are many different auction mechanisms that can be classified according to their features [29]. We ran experiments with all the implemented mechanisms under the hypotheses, that, according to the "Revenue Equivalence Theorem" (RET [39]), lead to the existence of an optimal bidder's strategy. The experiments demonstrated that RET is satisfied (up to some error due to discretisation), giving empirical evidence of the correctness of the implementation.

Many applications had also been developed using the ancestor of DCaseLP, CaseLP: a prototype of a multimedia, multichannel, personalised news provider, [19], was developed in collaboration with Ksolutions s.p.a. as part of the ClickWorld project, a research project partially funded by the Italian Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR). Older industrial applications involve freight train traffic [18] and vehicle monitoring [4].

The industrial applications of CaseLP and DCaseLP show an increased industrial interest and trust in both agent-based and declarative technologies, and demonstrate the liveliness of computational logic outside the boundaries of academia.

Verifying Interaction Protocols with Logic

We have recently developed a tool aimed at supporting verification of finite-state interaction protocols in a MAS setting, *West2East* [16], that exploits "*WE*b *S*ervice *T*echnologies to *E*ngineer *A*gent-based *S*of*T*ware" starting from the specification of an Agent Interaction Protocol (AIP). West2East exploits AUML [11] for representing AIPs, many different languages, including standard languages for Web Services, for sharing them, and Computational Logic to reason about them. In particular, West2East consists of a set of libraries for

1. *Translating visual AUML AIPs to various formats*: starting from an AUML interaction diagram graphically drawn using any UML editor, West2East generates the corresponding representation in many formats, including a Prolog term.

2. *Generating code compliant to the AIP*: starting from the Prolog term, a tuProlog program for each agent involved in the AIP is automatically generated by West2East. After a manual completion for adding the information missing in the AIP's specification, such as agents' state and guards of conditions, the tuProlog code can be run inside JADE thanks to the DCaseLP libraries.

3. *Reasoning about the AIP*: a mechanism for allowing tuProlog agents to reason about an AIP by exploiting meta-programming techniques is provided by West2East. Existential and universal properties, such

as "There is one path of the protocol where I will receive $message_1$", and "Whatever the path, I will send $message_2$", can be verified.

In [21] we have further investigated in the relation between (constraint) logic programming and infinite-state verification. More specifically, in [21] we show that a CLP bottom-up evaluation procedure can be applied to automatically verify safety and liveness properties for skeletons of communication protocols (with a fixed number of processes) like mutual-exclusion algorithms. In the case-studies described in [21] the source of infiniteness is the presence of potentially unbounded integer variables in the specification of individual processes. Constraints are used here to symbolically represent infinite collections of system configurations with a fixed number of processes.

Another interesting research line concerns with the application of linear logic programming to verification of infinite-state systems. Linear logic [28] is a suitable logical framework for the specification of concurrent systems. The LO fragment [3] of full linear logic provides multi-headed linear implications with only multiplicative disjunction and additive conjunction in the body. By exploiting and generalizing the connection between verification and logic programming described in [21], in [14] we have defined a bottom-up evaluation strategy for (first order) LO programs based on an effective fixpoint operator à-la T_P (the immediate consequence operator for (constraint) logic programs). The LO T_P operator works on first order multi-headed LO clauses [14]. Furthermore, it can be viewed as a symbolic predecessor operator for transition systems described via multiset rewriting systems defined over first-order atomic formulas. In [15] we have extended the bottom-up evaluation procedure to first order linear logic specification with universally quantified goals. In [13] we have applied the resulting procedure to verify properties of cryptographic protocols for any possible number of principals and parallel sessions.

Conclusions

Research on computational logic in Genova is very lively, and will be even more in the future thanks to the interest on its practical applications raised outside the boundaries of academia. Part of this research has been carried out in joint projects with the Logic Programming and Automated Reasoning Group in Torino. The results of these projects are described in [7, 5], and the active collaboration in witnessed by many other joint activities [8, 9].

The connections between the Logic Programming Groups in Torino and Genova date back to more than 30 years ago. The heads of the groups, Alberto and Maurizio Martelli, besides the same family name, share many common experiences: they worked together at the National Research Council in Pisa, were involved in the committees of conferences and workshops on Computational Logics, and, when moved to Torino and Genova respectively, founded research groups with the same objectives. The profitable collaboration will be pursued in the future with the hope to contribute in making research on Computational Logic an Italian excellence.

REFERENCES

[1] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49:672–713, 2002.

[2] D. Ancona and V. Mascardi. Coo-BDI: Extending the BDI model with cooperativity. In J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, *Proc. of the 1st Declarative Agent Languages and Technologies Int. Workshop, DALT'03, Revised Selected and Invited Papers*, LNAI, pages 109–134. Springer, 2004.

[3] J-M. Andreoli and R. Pareschi. Linear ojects: Logical processes with built-in inheritance. *New Generation Comput.*, 9(3/4):445–474, 1991.

[4] E. Appiani, M. Martelli, and V. Mascardi. A multi-agent approach to vehicle monitoring in motorway. Technical report, Computer Science Department of Genova University, 2000. DISI TR-00-13, Poster session of the Second European Workshop on Advanced Video-Based Surveillance Systems, AVBS 2001.

[5] M. Baldoni, C. Baroglio, G. Berio, A. Martelli, V. Patti, M. L. Sapino, C. Schifanella, M. Alberti, M. Gavanelli, E. Lamma, F. Riguzzi, S. Storari, F. Chesani, A. Ciampolini, P. Mello, M. Montali, P. Torroni, A. Bottrighi, L. Giordano, V. Gliozzi, G. L. Pozzato, D. Theseider Dupré, P. Terenziani, G. Casella, and V. Mascardi. Modeling, verifying and reasoning about web services. *Intelligenza Artificiale*. To appear.

[6] M. Baldoni, C. Baroglio, I. Gungui, A. Martelli, M. Martelli, V. Mascardi, V. Patti, and C. Schifanella. Reasoning about agents' interaction protocols inside DCaseLP. In J. A. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Proc. of the 2nd Declarative Agent Languages and Technologies Int. Workshop, DALT'04, Revised Selected and Invited Papers*, volume 3476 of *LNCS*, pages 112–131. Springer, 2004.

[7] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, C. Schifanella, L. Torasso, and V. Mascardi. Personalization, verification and conformance for logic-based communicating agents. In F. Corradini, F. De Paoli, E. Merelli, and A. Omicini, editors, *Proc. of the WOA 2005 National Workshop, Dagli Oggetti Agli Agenti*, pages 177–183. Pitagora Editrice Bologna, 2005.

[8] M. Baldoni, C. Baroglio, and V. Mascardi, editors. *Proceedings of the Multi-Agent Logics, Languages, and Organisations, Federated Workshops, MALLOW'007, Agent, Web Services and Ontologies, Integrated Methodologies (MALLOW-AWESOME'007) workshop, Durham, GB*. 2007.

[9] M. Baldoni, A. Boccalatte, F. De Paoli, M. Martelli, and V. Mascardi, editors. *WOA, Workshop dagli Oggetti agli Agenti, Proceedings*. Seneca Edizioni (Italy), 2007.

[10] M. Baldoni, L. Giordano, A. Martelli, and V. Patti. Modeling agents in a logic action language. In *Proc. of the Workshop on Practical Reasoning Agents, FAPR 2000*, 2000.

[11] B. Bauer, J. P. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent software systems. In P. Ciancarini and M. Wooldridge, editors, *Proc. of the 1st Agent-Oriented Software Engineering Int. Workshop, AOSE'00, Revised Papers*, volume 1957 of *LNCS*, pages 91–104. Springer, 2000.

[12] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE.* Wiley, 2007.

[13] M. Bozzano and G. Delzanno. Automatic verification of secrecy properties for linear logic specifications of cryptographic protocols. *J. Symb. Comput.*, 38(5):1375–1415, 2004.

[14] M. Bozzano, G. Delzanno, and M. Martelli. An effective fixpoint semantics for linear logic programs. *TPLP*, 2(1):85–122, 2002.

[15] M. Bozzano, G. Delzanno, and M. Martelli. Model checking linear logic specifications. *TPLP*, 4(5-6):573–619, 2004.

[16] G. Casella and V. Mascardi. West2East: exploiting WEb Service Technologies to Engineer Agent-based SofTware. *IJAOSE*, 1(3/4):396–434, 2007.

[17] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs*, pages 52–71, 1981.

[18] A. Cuppari, P. L. Guida, M. Martelli, V. Mascardi, and F. Zini. An agent-based prototype for freight trains traffic management. In P. G. Larsen, editor, *Proc. of the 5th FMERail Workshop. Held in conjunction with FM'99.* Springer, 1999.

[19] M. Delato, A. Martelli, M. Martelli, V. Mascardi, and A. Verri. A multimedia, multichannel and personalized news provider. In G. Ventre and R. Canonico, editors, *Proc. of the 1st Int. Workshop on Multimedia Interactive Protocols and Systems, MIPS 2003*, volume 2899 of *LNCS*, pages 388–399. Springer, 2003.

[20] G. Delzanno and M. Martelli. Proofs as computations in linear logic. *Theoretical Computer Science*, 258(1–2):269–297, 2001.

[21] G. Delzanno and A. Podelski. Constraint-based deductive model checking. *STTT*, 3(3):250–270, 2001.

[22] E. Denti, A. Omicini, and A. Ricci. Multi-paradigm Java-Prolog integration in tuProlog. *Sci. Comput. Program.*, 57(2):217–250, 2005.

[23] T. Eiter, V.S. Subrahmanian, and G. Pick. Heterogeneous active agents, I: Semantics. *Artificial Intelligence*, 108(1-2):179–255, 1999.

[24] E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

[25] M. Fisher and H. Barringer. Concurrent METATEM processes – A language for distributed AI. In *Proceedings of the European Simulation Multiconference*. SCS Press, Copenhagen, Denmark, 1991.

[26] E. Friedman-Hill. *Jess in Action : Java Rule-Based Systems (In Action series).* Manning Publications, 2002.

[27] G. De Giacomo, Y. Lespérance, and H. J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169, 2000.

[28] J-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[29] P. Klemperer. *Auctions: Theory and practice*. Princeton University Press, 2004.

[30] V. Mascardi, D. Briola, M. Martelli, R. Caccia, and C. Milani. Monitoring and diagnosing railway signalling with rule-based distributed agents. Technical report, Dipartimento di Informatica e Scienze dell'Informazione, University of Genova, Italy, 2008. Technical Report DISI-TR-08-04.

[31] V. Mascardi, M. Martelli, and I. Gungui. DCaseLP: a prototyping environment for multi-language agent systems. In M. Dastani, A. El-Fallah Seghrouchni, J. Leite, and P. Torroni, editors, *Proc. of the 1st Int. Workshop on Languages, Methodologies and Development Tools for Multi-Agent Systems, LADS'007*, LNCS. Springer, 2008. To appear.

[32] V. Mascardi, M. Martelli, and L. Sterling. Logic-based specification languages for intelligent software agents. *TPLP*, 4(4):429–494, 2004.

[33] R. Montagna, G. Delzanno, M. Martelli, and V. Mascardi. BDI^{ATL}: An alternating-time BDI logic for multiagent systems. In M. P. Gleizes, G. A. Kaminka, A. Nowé, S. Ossowski, K. Tuyls, and K. Verbeeck, editors, *Proc. of the 3rd European Workshop on Multi-Agent Systems, EUMAS'05*, pages 214–223. Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten, 2005.

[34] A. S. Rao and M. P. Georgeff. Asymmetry thesis and side-effect problems in linear-time and branching-time intention logics. In J. Myopoulos and R. Reiter, editors, *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence, IJCAI-91*. Morgan Kaufmann publishers, 1991.

[35] A. S. Rao and M. P. Georgeff. Deliberation and intentions. In *Proc. of 7th Conference on Uncertainity in Artificial Intelligence*. Morgan Kaufmann publishers, 1991.

[36] A. S. Rao and M. P. Georgeff. Modelling rational agents within a BDI-architecture. In *Proc. of the 2nd Int. Conference of Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann publishers, 1991.

[37] D. Roggero, F. Patrone, and V. Mascardi. Designing and implementing electronic auctions in a multiagent system environment. In F. Corradini, F. De Paoli, E. Merelli, and A. Omicini, editors, *Proc. of the WOA 2005 National Workshop, Dagli Oggetti Agli Agenti*, pages 157–163. Pitagora Editrice Bologna, 2005.

[38] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[39] W. Vickrey. Auction and bidding games. In *Recent advances in Game Theory*, pages 15–27. Princeton University Conference, 1962.

[40] M. Wooldridge. *Reasoning about rational agents.* Mit press, 2000.

PROGRAMMAZIONE LOGICA IN DATALOG: UN LUNGO PERCORSO DALLA TEORIA ALLA PRATICA

LOGIC PROGRAMMING IN DATALOG: A LONG TOUR FROM THEORY TO PRACTICE

Sergio Greco, Luigi Palopoli, Nicola Leone, Pasquale Rullo, Domenico Saccà

SOMMARIO/*ABSTRACT*

In questo articolo si descrivono le linee di ricerca sviluppate a Cosenza nell'ambito della programmazione logica in un arco temporale di oltre 20 anni e che hanno portato a recenti interessanti e promettenti sviluppi industriali. Tali linee di ricerca sono cambiate nel tempo ma hanno mantenuto l'interesse iniziale per accoppiare la programmazione logica con le tecnologie della basi di dati, interesse che si continuamente rinnovato per affrontare nuove sfide nell'uso della teoria per risolvere problemi pratici.

In this paper, we describe the research lines in logic programming, carried out in Cosenza over a period of more than 20 years, which have recently produced promising industrial exploitation follow-ups. The research lines have changed over the time but they have kept the initial interest on combining logic programming with databases techniques, that has been continuously renewed to cope with new challenges, in our attempt to use theory to solve practical problems.

Keywords: Logic programming, DATALOG, bottom-up execution, stable models, disjunctive logic programming, answer set programming, ontology

1 Introduction

The research on logic programming in Cosenza started in the middle eighties at CRAI, an industrial consortium for information technology research and applications at Rende, Italy. With the enthusiasm of young researchers, we decided to enlarge our original competencies in database technology to exploit the great, at that time promising potentialities of logic programming. We then set up a research group on DATALOG (a version of logic programming particularly suited for database applications) and we started to study the problem of the efficient computation of answers to logic queries over relational databases.

Later on, while moving from CRAI to University of Calabria, we also moved to more theoretical issues in DATALOG. In particular, we worked at endowing DATALOG with the capability of handling non monotonic reasoning and defeasible knowledge, and we concentrated on providing a non-classical interpretation for negation and disjunction (with the two perspectives not necessarily disjoint).

When the age of maturity came, our interest for theory was more and more urged to be combined with the necessity of providing evidence of its utility by means of "running" prototypes. So, our group promoted the construction of the DLV system, one of the most efficient implementation of logic programming available to date, which is being used in many applications.

Finally, getting old and desiring to leave a more tangible effect of our research to the economy of our Region, along with other researchers from University of Calabria, we founded a research spin-off, named Exeura, whose mission is to transform research results in the field of Knowledge Management into industrial products.

In this paper we make a quick tour of our research in DATALOG during the last 20 years, starting from the infancy of our work on efficient query compilation, to the youth of the contribution on non monotonic reasoning and the maturity of developing the DLV system, eventually arriving to the old age of exploiting results within an industrial framework. But our story does not end here: we are positive and ready to add another chapter!

2 INFANCY: efficient compilation of DATALOG

DATALOG is essentially logic programming without function symbols using tuples of a relational database as facts: a database $\mathcal{D}$ is seen as a set of facts , whose predicate symbols (*extensional predicates*) coincide with the relation names, and all other predicate symbols (*intensional predicates*), defined by rules, correspond to views of the database.

We have investigated the efficient computation of answers to logic queries over relational databases since the the middle of eighties of the past century when we were all working for CRAI. The research has concentrated firstly on the definition of algorithms for the efficient computation of the semantics of programs and answers. The problem can be stated as follows: given a query $\mathcal{Q} = \langle q(X), \mathcal{P} \rangle$ and a database $\mathcal{D}$, seen as a set of facts, compute the atoms matching $q(X)$ which are logic consequence of $\mathcal{P} \cup \mathcal{D}$. Two main approaches have been proposed in the literature, known as *top-down* computation (used by Prolog-like system) and *bottom-up* computation (used by deductive database and answer-set systems).

The advantages of top-down systems is that only rules and atoms relevant to the query goal are considered, but there are several problems as termination and duplicated computation. On the other side, the bottom-up strategy always terminates, but it computes atoms which are not relevant for the query goal (i.e. first computes all atoms which are logic consequence of $\mathcal{P} \cup \mathcal{D}$ and, afterwards, selects the atoms matching the goal $q(X)$). Concerning the definition of optimization techniques avoiding duplicated computation, the most important contributions were top-down methods with memoing strategy and the semi-naive (bottom-up) algorithm.

Moreover, several optimization techniques combining top-down and bottom-up strategies were proposed as well. These techniques try to compute only atoms which may be "relevant" for the query goal in a bottom-up strategy. The key idea of all these techniques consists in the rewriting of deductive rules with respect to the query goal so that answering the query without actually computing irrelevant facts. General rewriting techniques (e.g. magic-set and supplementary magic set) can be applied to all queries, but their efficiency is limited. On the other side, there are specialized techniques which are very efficient, but they can be applied to limited classes of queries.

We investigated a particular interesting class of queries, known as *chain queries*, i.e., queries where bindings are propagated from arguments in the head to arguments in the tail of the rules, in a chain-like fashion. For these queries, which are rather frequent in practice (e.g., graph applications), insisting on general optimization methods (e.g., the *magic-set* method) does not allow to take advantage of the chain structure, thus resulting in rather inefficient query executions. Specialized methods for subclasses of chain queries have been proposed in the literature, but, unfortunately, these methods do not fully exploit bindings.

We proposed a *counting* method that is particularly specialized for bound chain queries; however this method, although proposed in the context of general queries [Saccà(13,17,20), Greco(8)], preserves the original simplicity and efficiency only for a subset of chain queries whose recursive rules are linear. We later proposed a new method exploiting the relationship between chain queries, context-free languages and pushdown automata, which permits to rewrite queries into a format that is more suitable for the bottom-up evaluation [Greco(18, 43)]. The so-called *pushdown* method translates a chain query into a factorized left-linear program implementing the pushdown automaton recognizing the language associated with the query. A nice property of this method is that it reduces to the counting method in all cases where the latter method behaves efficiently and introduces a unified framework for the treatment of special cases, such as the factorization of right-, left-, mixed-linear programs, as well as the linearization of non-linear programs.

These techniques defined for standard DATALOG queries can be also applied, or easily extended, to disjunctive logic queries (queries whose associated program is a disjunctive DATALOG program) [Greco(36, 87, 110), Leone(144)].

We also elaborated optimization techniques for queries with aggregates (expressing, for instance, optimization problems) [Greco(15,38,45,60,75)]. These techniques rewrite queries so that the simple modification of the semi-naive algorithm emulates classical optimization strategies such as greedy and dynamic programming.

Further on, we investigated optimizations techniques for queries with complex terms such as sets and, in particular, we analyzed the computation of optimal sets of matchers and unifiers for atoms with "bounded" set terms [Greco(21,24)].

3 YOUTH: non monotonic reasoning in DATALOG

The realization of common-sense reasoning systems has been, since the beginning, one of the natural application realm of logic programming. However, common-sense reasoning requires non-monotonicity, that is, the capability for the reasoning system to cancel or retract previously attained conclusions, in the light of new evidence the system becomes aware of (that is, generally, the knowledge must be defeasible). Unfortunately, plain DATALOG is indeed monotonic and, therefore, unapt to the purpose. Therefore, mechanisms had to be devised in order to endow DATALOG-based languages with the capability of representing and managing non-monotonicity.

Loosely speaking, in order to endow DATALOG with the capability of handling defeasible knowledge, one might resort to non standard semantics for the language as a whole or concentrate on providing a non-classical interpretation for negation or disjunction (with the two perspectives not necessarily disjoint). In the logic programming community the second line of research received a much larger deal of attention than the first one, and we followed this lines in the research developed when all of us moved from CRAI to University of Calabria.

Important results derived by the work developed in our group on ordered logic programs, that are, programs consisting of a poset of modules, each of which is itself a logic program, under the assumption that programs lying higher in the hierarchy are semantically more trustable than lower ones [Leone(6), Rullo(29)] and the related issue of inheritance [Leone(91), Rullo(18)] and, moreover, on the circumscription-based interpretation of negation-free DATALOG [Palopoli(32)]. The semantics of (disjunctive) logic programs with preferences on atoms was later investigated in [Greco(136)].

But we also investigated the semantics of negation in DATALOG-like languages. The simpler form of non-classical interpreting negation in DATALOG is encoded in the notion of *stratified* (aka, *perfect*) models due, among others, to Przyimusinski. This semantics is defined when in a program there is no recursion through negation, that is, the program is *stratified*. In this case, one can divide the program in an ordered list of layers, such that each predicate occurring negated in the rule of any layer does not occur in the head of rules of that and higher layers. Then, the intended model is obtained by evaluating the program layer-wise, beginning with the lowest one. The drawback of stratified programs rests on their limited generality and expressive power. A more general notion is that of *locally stratified* programs, where recursion through negation is allowed as long as it gets resolved at the ground level. Contrary to stratification, however, local stratification is in general undecidable, even if sufficient conditions for it can be given [Palopoli(3)].

Van Gelder and others proposed a solution to the problem of providing a clean semantics to programs with recursive negation, by defining the concept of *well-founded model*. Loosely speaking, a model of a DATALOG program is well-founded is the model does not contain any subset of unjustifiable atoms. Well-founded semantics is in general polynomial-time computable, but its implementation is not at all trivial [Rullo(9)]. Also, the well-founded model of a DATALOG program is unique (but may not exist).

In order to attain a significant boost in expressivity, though, one has to consider a further, simple yet powerful, semantics for logic program with negation, namely, that of *stable models* of Gelfond and Lifschitz. To informally illustrate, a model of a program is stable if the program regenerates it when the knowledge encoded in the model is assumed from granted. In general, a DATALOG program may have none, one or multiple stable models. Therefore, differently from the aforementioned semantics, entailment under stable model semantics can be intuitively defined in two forms, that are, cautious reasoning, which tells an information to be implied by a program if it is indeed implied in all the stable models of the program, and brave reasoning, which makes the information implied by the program if implied by at least one of its stable models. These definitions endow DATALOG programs with a much larger expressiveness (allowing to capture classes like coNP) than that of well-founded models but, at the same time, renders the entailment and related problem intractable [Saccá(49)], thereof including that of computing one single stable model of a DATALOG program. Fortunately, there are indeed cases when the stable model semantics can be computed efficiently [Palopoli(22)].

But besides the cautious and the brave forms of reasoning, the possibility for a program to have multiple stable models can be interpreted in a different and rather appealing manner, that is, that each stable model of the program *non-deterministically* encode one possible status of the world. This view of stable model semantics prompted through years some of us to study the formal properties and the potential application of exploiting non-determinism as encoded in DATALOG programs under the stable model semantics, which resulted in several interesting research papers [Greco(28,29), Greco(44), Saccá(43,48)].

A second depart from more standard forms of semantics is determined by allowing more than two truth values for literals. To illustrate, in two-valued semantics, each literal of a program must be either declared true or false. There are cases and applications, though, where it appears sensible to introduce a third truth value, say ”unknown”, into play, to be assigned to a literal if neither itself nor its negation is entailed by a set of rules. All that is conducive to the notion of *partial* model, which is precisely one that tells some atoms true, some atoms false and some undefined in the status of the world it encodes for. Also in this new setting it is worth analyzing the formal properties of the resulting formalisms [Saccá(58)]. Moreover, in this setting, stable models semantics also allows to express search and optimization problems [Saccá(54,63,83)].

To suitably deal with negation is not enough for some application though. Theoretically speaking, this happens when one deals with problems which are complete for the second level of the polynomial hierarchy. From the more practical viewpoint, this more simply happens when the application context naturally calls for the exploitation of disjunctive statements, that are, statements which declare the (possibly conditional) truth of at least one of a group of atoms. The resulting language, usually referred to as *Disjunctive DATALOG* allows disjunct to occur in rule heads and (possibly) negation in rules bodies. Several of the issues discussed above for (disjuntion-free) DATALOG carry over to Disjunctive DATALOG, and the development of the associated research lines has witnessed a relevant contribution of our group. To illustrate, the papers [Rullo(22), Greco(98)] include fundamental results about the semantics, complexity and expressive power of Disjunctive DATALOG programs, [Palopoli(27,50)] discuss tractability issues about Disjunctive DATALOG, while [Leone(103,113)] tackles with the issue of selecting some of the models of a Disjunctive program as the preferred ones and, finally, [Rullo(33)] deals with the enhancement of the expressive and representational capabilities of Disjunctive DATALOG using constraints.

4 MATURITY: the DLV system

DLV [Leone(139)] is an advanced system for Knowledge Representation and Reasoning which is based on Disjunctive DATALOG under the stable model semantics (also called Answer Set Programming). Roughly, a Disjunctive DATALOG program is a set of disjunctive rules, i.e., clauses of the form

$$a_1 \mathtt{v} \cdots \mathtt{v} a_n \mathtt{:-}\ b_1, \cdots, b_k, \mathtt{not}\ b_{k+1}, \cdots, \mathtt{not}\ b_m$$

where atoms $a_1, \ldots, a_n, b_1, \ldots, b_m$ may contain variables. The intuitive reading of such a rule is "If all $b_1, \ldots, b_k$ are true and none of $b_{k+1}, \ldots, b_m$ is true, then at least one atom in $a_1, \ldots, a_n$ must be true." Disjunctive DATALOG has a very high expressive power – it allows to express all problems in the complexity class Σ_2^P (i.e., NP^{NP}). Thus, under usual complexity conjuctures, Disjunctive DATALOG is strictly more expressive than both SAT and CSP, the power of which is "limited" to NP, and it can naturally represent a large class of relevant problems ranging from artificial intelligence to advanced database applications.

DLV is generally considered the state-of-the-art implementation of Disjunctive DATALOG. Its efficiency has been confirmed by the results of First Answer Set Programming System Competition (http://asparagus.cs.uni-potsdam.de/contest/), where DLV won the "disjunctive" category. Moreover, DLV turned out to be very efficient also on (non-disjunctive) DATALOG programs, as it finished first also in the general category MGS (Modeling, Grounding, Solving – also called *royal* competition, open to all ASP systems).

The implementation of the DLV system is based on very solid theoretical foundations, and exploits major results that have been achieved by the Deductive Databases group of University of Calabria in the last 20 years. The system has been recently engineered for industrial exploitation, and is successfully employed in many challenging real-world applications, for instance in the area of Knowledge Management [Leone(141)], and advanced Information Integration [Leone(127,144)] (see next section).

Among the many features of the system, it is worth remarking the following:

Advanced knowledge modeling capabilities. DLV provides support for declarative problem solving in several respects:

- High expressiveness in a formally precise sense (Σ_2^P), so any such problem can be uniformly solved by a fixed program over varying input.

- Rich language for knowledge modeling, extending Disjunctive DATALOG with weak constraints (for preferences handling) [Leone(68)], powerful aggregate functions [Leone(132,110,124,148)], and other useful KR constructs.

- Full declarativeness: ordering of rules and subgoal is immaterial, the computation is sound and complete, and its termination is always guaranteed.

- Declarative problem solving following a "Guess&Check" paradigm [Leone(139)] where a solution to a problem is guessed by one part of a program and then verified through another part of the program.

- A number of front-ends for dealing with specific AI applications [Leone(57,109,105,125)], information extraction [Leone(141)], Ontology Representation and Reasoning [Leone(146,130)].

Solid Implementation. Much effort has been spent on sophisticated algorithms and techniques for improving the performance, including

- Database optimization techniques: indexing, join ordering methods [Leone(85)], Magic Sets [Leone(144,124)].

- Artificial intelligence computation techniques: heuristics [Leone(87,149,133)], backjumping techniques [Leone(138,119)], pruning operators [Leone(137)].

DLV is able to solve complex problems and efficiently deal also with large input data [Leone(147)].

Database Interfaces. The DLV system provides a general ODBC interface to relational database management systems [Leone(120)].

For up-to-date information on the system and a full manual we refer to http://www.dlvsystem.com, where also download binaries of the current release and various examples are available.

5 OLD AGE: industrial applications of DLV

We are finally at the end of the story. In 2002, along with other researchers from University of Calabria, we founded a research spin-off, named Exeura. Since the beginning, the mission of Exeura was to transform into commercial products research results in the field of Knowledge Management (KM). Topics of interest include: (1) Knowledge Representation and Reasoning (e.g., ontologies, automatic reasoning, etc.); (2) Data, Text, and Process Mining (e.g., data discovery in databases, document classification, web mining, workflow mining, etc.); (3) Information Extraction and Wrapping; (4) Heterogeneous Information Sources Integration.

Thanks to a vast scientific and technological know how in KM and, in general, in advanced Information Systems, Exeura has implemented a number of industrial prototypes, currently under productization. Some of those exploit the DLV reasoning capabilities, notably, OntoDLV, Olex and Hylex.

OntoDLV is a system for ontology specification and reasoning [Leone(146)]. Ontologies are abstract models of complex domains that have been recognized to be a fundamental tool for conceptualizing business enterprise information. The World Wide Web Consortium (W3C) has already provided recommendations and standards related to ontologies, like RDF(S) and OWL. In particular, OWL has been conceived for the Semantic Web, with the goal to enrich Web pages with machine-understandable descriptions of the presented contents. OWL is based on expressive Description Logics (DL); distinguishing features of its semantics w.r.t. Logic Programming languages are the adoption of the Open World Assumption (OWA) and the non-uniqueness of names (different names can denote the same individual). However, while the semantic assumptions of OWL make sense for the Web, they are unsuited for enterprise ontologies. Since an enterprise ontology describes the knowledge of specific aspects of the closed world of the enterprise, it turns out that the Closed World Assumption (CWA) is more appropriate than the OWA (appropriate for the Web, which is an open domain). Moreover, the presence of naming conventions, often adopted in enterprises, can guarantee name uniqueness, making also the Unique Name Assumption (UNA) plausible. Importantly, enterprise ontologies often are the evolution of relational databases, where both CWA and UNA are mandatory.

OntoDLV supports a powerful ontology representation language, called OntoDLP, extending (disjunctive) Answer Set Programming (ASP) with all the main ontology features including classes, inheritance, relations and axioms. OntoDLP is strongly typed, and includes also complex type constructors, like lists and sets. The semantic peculiarities of ASP, like the Closed World Assumption (CWA) and the Unique Name Assumption (UNA), allow to overcome both the above mentioned limits of OWL, thus making OntoDLV suitable for enterprise ontology specification. It is worth noticing that OntoDLV supports a powerful interoperability mechanism with OWL, allowing the user to retrieve information from OWL ontologies, and build rule-based reasoning on top of OWL ontologies. The system is already used in a number of real-world applications including agent-based systems, information extraction, and text classification.

Olex is a rule-based text classification system [Rullo(43)]. It supports a hypothesis language of the form

$$c \leftarrow T_1 \in d \vee \cdots \vee T_n \in d \wedge \neg(T_{n+1} \in d \vee \cdots \vee T_{n+m} \in d)$$

where each T_i is a conjunction of terms (n-grams). The meaning of a classifier as above is "classify document d under category c if any of $T_1, \cdots, T_n$ occurs in d and none of $T_{n+1}, \cdots, T_{n+m}$ occurs in d". The execution of a classifier relies on the DLV system.

One important feature of the Olex system is the integration of the manual approach with the automatic rule induction. Thanks to the interpretability of the produced classifiers, the domain expert can participate in the refinement of a classifier, by manually specifying a set of rules to be used in conjunction with those automatically learned. This co-operation, in fact, may be very effective, since both approaches have some limits, that can be overcome if used in synergy. To this end, the expressive power of the DLV language turns out of great advantage, as it allows the (manual) specification of complex classification rules (not restricted to the hypothesis language), e.g., rules with aggregate functions (such as *count*, *sum*, etc.) that are very useful in text categorization.

Olex has been applied to a number of real world applications in various industries including: health-care, tourism, and insurance.

HiLeX supports a semantic-aware approach to information extraction from unstructured data (i.e., documents in several formats, e.g., html, txt, doc, pdf, etc), that is currently used in many applications of Text Analytics. The semantic approach of HiLeX basically relies on [Sacc(115)]:

- the ontology representation formalism OntoDLP used for describing the knowledge domain;
- a logic-based pattern matching language relying on a two-dimensional document representation; in fact, a document is viewed as a Cartesian plane composed by a set of nested rectangular regions called portions. The DLV system is used as the pattern recognition engine.

REFERENCES

[1] Greco: Papers in the DBLP entry of Sergio Greco: 8, 15, 18, 21, 24, 28, 29, 36, 38, 43, 44, 45, 60, 75, 87, 98, 110, 136.

[2] Leone: Papers in the DBLP entry of Nicola Leone: 6, 57, 68, 85, 87, 91, 103, 105, 109, 110, 113, 119, 120, 124, 125, 127, 130, 132, 133, 137, 138, 139, 141, 144, 146, 147, 148, 149.

[3] Palopoli: Papers in the DBLP entry of Luigi Palopoli: 3, 22, 27, 32, 50.

[4] Rullo: Papers in the DBLP entry of Pasquale Rullo: 9, 18, 22, 29, 33, 43.

[5] Sacca: Papers in the DBLP entry of Domenico Saccà: 13, 17, 20, 43, 48, 49, 58, 115.

NOTE: *for the sake of space, citations are given referring to the DBLP bibliography* (http://www.informatik.uni-trier.de/∼ley/db/index.html), *using the format* [author(n)] *to mean the reference n of the DBLP bibliography list of author.*

6 Contacts

Sergio Greco, Luigi Palopoli, Domenico Saccà
DEIS Department, University of Calabria
Via P. Bucci 41/C, 87036 Rende (CS), Italy
Exeura s.r.l., Rende - Italy
{greco,palopoli,sacca}@deis.unical.it

Nicola Leone, Pasquale Rullo
Department of Mathematics, University of Calabria
Via P. Bucci 30/B, 87036 Rende (CS), Italy
Exeura s.r.l., Rende - Italy
{leone,rullo}@mat.unical.it

7 Biography

- Sergio Greco is full professor of Computer Engineering at the DEIS (Electronics, Computer and Systems Sciences) Department of the University of Calabria. His present research interests are: logic programming for knowledge representation and reasoning, imprecise answering systems, data mining, XML.
- Nicola Leone is full professor of Computer Science at the Department of Mathematics of the University of Calabria. His present research interests are: Knowledge Representation and Reasoning, Logic Programming and NonMonotonic Reasoning, Logics in Databases, Computational Complexity in Artificial Intelligence and Databases
- Luigi Palopoli is full professor of Computer Engineering at the DEIS (Electronics, Computer and Systems Sciences) Department of the University of Calabria. His present research interests are: bioinformatics, computational game theory, knowledge representation and data mining.
- Pasquale Rullo is full professor of Computer Science at the Department of Mathematics of the University of Calabria. His present research interests are: Data and Text Mining, Knowledge Representation
- Domenico Saccà is full professor of Computer Engineering at the DEIS (Electronics, Computer and Systems Sciences) Department of the University of Calabria. His present research interests are: scheme integration in data warehousing, compressed representation of datacubes, workflow and process mining, logic-based query languages for data mining and information extraction.

Modello e Verifica di Processi di Business e Coreografie in ALP

Modeling and Verification of Business Processes and Choreographies in ALP

Federico Chesani Paola Mello Marco Montali Paolo Torroni

SOMMARIO/*ABSTRACT*

Questo articolo introduce brevemente le nostre recenti attività di ricerca in relazione all'uso della programmazione logica per la specifica e verifica delle interazioni in vari contesti. L'articolo evidenzia alcuni risultati riportati in vari ambiti: sistemi multi-agente, servizi web e argomentazione, con particolare enfasi sugli aspetti collegati ai processi di business e alle coreografie di servizi Web.

In this article we overview our recent research activity concerning the use of logic programming for interaction specification and verification in several domains. We outline relevant results in the areas of multi-agent systems, argumentation and web services, and we devote a special emphasis to issues related to business processes and Web service choreographies.

Keywords: Logic programming, hypothetical reasoning, interaction, modelling, verification, multi-agent systems, protocols, business processes, web services, choreographies, semantic web, argumentation.

1 Background

Over two decades ago, a significant part of the Italian and European CS research community and IT industry expressed great interest in the new Logic Programming (LP) paradigm. Such an interest was sensibly encouraged by the pioneering work of Giorgio Levi, Franco Turini, Ugo Montanari, Alberto Martelli, and others. The Artificial Intelligence group of DEIS, School of Engineering, University of Bologna, was born at the time of the LP wave of the Eighties. Our group was attracted by the unique features of LP, including its ability to marry formal and practical aspects, and to enable the correspondence of a declarative language with an underlying execution model.

The main research directions of our group back then were centered around distribution, modularity, parallelism, and language extensions such as Constraint and Abductive Logic Programming. In order to enable programming in the large in the LP paradigm, two approaches have been studied for structuring logic programs: an algebraic method based on meta-operators, and another approach based on language extensions. The first model brought to the definition of an extended LP language called StructuredProlog [16], while the second approach was based on the introduction of negation in LP, to support non-monotonic reasoning.

StructuredProlog allows to to integrate blocks, modules, hypothetical reasoning, logical theory and object taxonomies. It has been implemented as an extension of the Warren Abstract Machine, via software emulation and then in hardware, and optimized using partial evaluation techniques. Past research also focussed on parallel logic languages with AND parallelism and no variable sharing on a MIMD architecture. Inter-process communication and synchronization was possible via multi-headed clauses and a shared blackboard, and an optimized unification mechanism specifically tailored to serve the purpose. Finally, the LP paradigm has been integrated with the OO paradigm, to define the Distributed Logic Objects language (DLO). In DLO, methods are expressed via multi-headed clauses, in a purely declarative style, while specific constructs are defined to express interaction among objects and inheritance.

2 The SOCS Project

Since 2001, the group has devoted most of its resources to the study of computational logic-based multi-agent systems [19], specifically agent interaction: the aim was to develop an LP-based language and an operational model for the specification and verification of agent interaction protocols. Such work has been carried out in the context of the EU-funded SOCS project [1] . The SOCS society model

[1]Project IST-2001-32530, 5FP. SOCS stands for "Societies Of ComputeeS: a computational logic model for the description, analysis and ver-

[21, 3], developed by a joint effort between the University of Bologna and the University of Ferrara, gives concrete guidelines for the formal specification of the interaction among agents that form a society, and for the definition of a computational logic-based architecture for agent interaction. In the proposed architecture, the society defines the allowed interaction protocols, which in turn are defined by means of *Social Integrity Constraints* (ICs). The society knowledge is defined as an abductive logic program [9]: ICs are used in order to express constraints on the communication patterns, while expected communicative acts ("*expectations*") are expressed as abducible predicates. Both the specification language and the underlying proof-procedure are called $\mathcal{S}$CIFF.

Expectations, whose intuition recalls the usual deontic operators of permission, obligation, and prohibition [8], are used to provide a semantics to both agent communication languages and to interaction protocols [6]. The resulting model is based on a declarative (logic) representation, therefore easy to understand.Moreover, its operational model can be exploited to achieve an implementation of societies of computees based on their formal specifications [2]. Thanks to the link between formal specification and implementation, the model provides also a good ground for the automatic verification and formal proof of properties [10].

The society model and the $\mathcal{S}$CIFF operational model were satisfactorily tested on a number of applications. These include resource exchange [11], e-commerce protocols [7], and combinatorial auctions [1]. A repository of protocols specified using $\mathcal{S}$CIFF is publicly available through the project's home page [22].

The SOCS-SI tool [4]) supports $\mathcal{S}$CIFF models and have been used for extensive experimentation. It takes as input the declarative formalisation, and it allows the automated verification of the social aspects of a SOCS application. SOCS-SI is general in its scope, and has been interfaced to other implemented agent platforms, such as JADE, and to other non-agent related communication platforms, like e.g. TuCSoN. SOCS-SI uses the $\mathcal{S}$CIFF proof procedure, that has been implemented using SICStus Prolog, and in particular its CHR library. The interested reader can learn more about $\mathcal{S}$CIFF in [5], and in the tutorial paper [17]. SOCS-SI and $\mathcal{S}$CIFF are publicly available on the web [2] .

3 Current Research Directions

Most of our current research has originated from the outcomes of SOCS. Starting form the many analogies between the agent paradigm and the Web service model, interaction protocols and choreographies has been the subject of conspicuous research carried out in the context of two recent national projects lead by Alberto Martelli [3] . Part of the research activity done within these projects has built on $\mathcal{S}$CIFF to i) produce new formalisms for the specification and verification of interaction protocols and choreographies; and to ii) develop new techniques for automatic property verification and reasoning about Web Services.

The translation of graphical modeling languages into the formal languages developed in these projects has been also subject of research. Our group has studied the translation of choreographies (represented in WS-CDL or in BPMN) into its corresponding $\mathcal{S}$CIFF specification, focussing on verification of compliance. Several tools, based on the $\mathcal{S}$CIFF procedure, have been developed to cope with complete logs and with run-time events. Further supported types of verification regard the proof of "high level" properties, such as verifying in an e-commerce scenario, that a buyer is guaranteed to receive the good he/she paid for, and the seller is guaranteed to be paid.

Alongside our research on Web Services, we have extended and applied $\mathcal{S}$CIFF in the context of agent-oriented requirements engineering. This has brought to the development of $\mathcal{B}$-Tropos ($\mathcal{B}$ standing for *Business*): a unified framework for information systems engineering, with the aim to reconcile requirements elicitation with declarative specification, prototyping, and analysis [15]. $\mathcal{B}$-Tropos, built on the well-known Tropos methodology [14], lets the user to express temporal and data constraints between tasks, hence introducing also the concepts of start and completion times, triggering events, and deadlines. The verification capabilities supported by the $\mathcal{S}$CIFF proof allow prototyping (animation) and analysis (properties and conformance verification) directly in $\mathcal{B}$-Tropos. Early requirements engineers will be able to test their models directly; engineers testing model properties will not have to resort to ad-hoc, error-prone translations of high-level models into other languages, thanks to the automatic translation of $\mathcal{B}$-Tropos models into $\mathcal{S}$CIFF programs; finally, managers monitoring the correct behavior of a system will exploit the $\mathcal{S}$CIFF specification to check the compliance using the SOCS-SI runtime and off-line checking facilities [4].

Another current research direction which builds on SCIFF concerns argumentation in the Semantic Web [24]. Our work resulted in the development of an operational argumentation framework, called ArgSCIFF, to support dialogic argument exchange between Semantic Web Services. In ArgSCIFF, an intelligent agent can interact with a Web Service and reason from the interaction result. The reasoning semantics is an argumentation semantics that views the interaction as a dialogue. The dialogue lets two parties exchange arguments and attack, challenge, and justify them

ification of global and open societies of heterogeneous computees." The project run for 42 months, from January 2001 until June 2005, and it involved 6 academic institutions, including the University of Bologna and the University of Ferrara. See [12, 23].

[2] `http://lia.deis.unibo.it/research/socs_si` and `http://lia.deis.unibo.it/research/sciff`

[3] In 2004-2005, our group has been involved as a partner in the National MIUR (ex 40%) project on "Development and verification of logic-based multi-agent systems," and in 2006-2007 on the National PRIN (ex 40%) project on "Specification and verification of agent interaction protocols." For more information, see the project Web site [20]. A report on the most recent project is due to appear on this magazine [13].

on the basis of their knowledge. This format has the potential to overcome a well-known barrier to human users adoption of IT solutions because it permits interaction that includes justified answers that can be reasoned about and rebutted.

4 CLIMB

Actually, a great deal of our resources are devoted to the development of LP-based techniques for modeling and verifying business processes and choreographies. The reference framework for this work is called CLIMB [4] . As specification language, CLIMB adopts an extension of DecSerFlow/Condec, a family of graphical languages for the declarative specification of service/business flows [26]. Graphical models are then automatically mapped onto $\mathcal{S}$CIFF, integrating the best of the two approaches:

- CLIMB models are declarative and open. They do not specify one particular flow of execution, but rather focus on the set of constraints that must be satisfied by interacting entities. Constraints specify either what is mandatory or forbidden during execution.
- Different verification tasks can be applied on CLIMB models by exploiting the proof-theoretic operational counterpart of $\mathcal{S}$CIFF as well as different logic programming techniques.

In particular, CLIMB exploits $\mathcal{S}$CIFF for carrying out both run-time and a-priori verification tasks.

At run-time, $\mathcal{S}$CIFF can be used as an alerting infrastructure capable to perform *compliance checking*, i.e., verifying whether a concrete process execution (or service interaction) complies with the prescribed model (and detecting violations as soon as possible). Such a verification can be seamlessly applied a-posteriori as well, checking already completed execution traces. In this respect, CLIMB rules are used as an intuitive classification criterion which split analyzed traces into a compliant and non compliant sub-sets; a plug-in which exploits such a reasoning technique has been implemented and integrated inside the ProM[25] process mining framework.

At static time, the "generative" variant of the $\mathcal{S}$CIFF proof procedure can be exploited to check the consistency of developed models, by detecting the presence of conflicts (which undermine the possibility of executing the model) and by discovering if they contain dead activities (i.e., activities that can be never executed). Such verifications constitute the basis also for determining if different CLIMB models can be composed without introducing conflicts. This is particularly important in a service-oriented setting, where a choreography can be intended as a contract aiming to make different partners correctly collaborate, and then a set of compatible concrete services implementation must be found to concretely implement the system.

It is worth noting that DecSerFlow/Condec models have an alternative underlying semantics in terms of Linear Temporal Logic formulas, which enable the possibility to apply model checking techniques in order to verify the designed models. In this respect, a research activity focused on more foundational aspects is being carried out, to compare expressivity, complexity and reasoning capabilities of the two frameworks.

[4]CLIMB stands for "Computational Logic for the verIfication and Modeling of Business processes and choreographies." For more information and up-to-date resources the interested reader can refer to the CLIMB Web site [18].

Acknowledgements

Much of the work presented here has been done in tight cooperation with the AI group of the University of Ferrara. This paper is complementary to [12], where they focus on the learning and property verification issues in relation with the work carried out within and following SOCS.

REFERENCES

[1] M. Alberti, F. Chesani, M. Gavanelli, A. Guerri, E. Lamma, P. Mello, and P. Torroni. Expressing interaction in combinatorial auction through social integrity constraints. *Intelligenza Artificiale*, II(1):22–29, 2005.

[2] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. A logic based approach to interaction design in open multi-agent systems. In *Proc. 13th IEEE international Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WET-ICE 2004)*, pages 387–392. IEEE Press, 2004.

[3] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. The SOCS computational logic approach for the specification and verification of agent societies. In *Global Computing*, volume 3267 of *LNAI*, pages 324–339. Springer-Verlag, 2005.

[4] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Compliance verification of agent interaction: a logic-based tool. *Applied Artificial Intelligence*, 20(2-4):133–157, Feb.-Apr. 2006.

[5] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Transactions on Computational Logic*, 9(4), 2008.

[6] M. Alberti, A. Ciampolini, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. A social ACL semantics by deontic constraints. In *Multi-Agent Systems and*

Applications III, volume 2691 of *LNAI*, pages 204–213. Springer-Verlag, 2003.

[7] M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. Specification and verification of agent interaction protocols in a logic-based system. In *Proceedings of the 19th Annual ACM Symposium on Applied Computing (SAC 2004)*, pages 72–78. ACM Press, 2004.

[8] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, G. Sartor, and P. Torroni. Mapping deontic operators to abductive expectations. *Computational and Mathematical Organization Theory*, 12(2-3):205–225, Oct. 2006.

[9] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. An Abductive Interpretation for Open Societies. In *Advances in Artificial Intelligence*, volume 2829 of *LNAI*, pages 287–299. Springer-Verlag, 2003.

[10] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interactions using social integrity constraints. *ENTCS*, 85(2), 2003.

[11] M. Alberti, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Modeling interactions using *Social Integrity Constraints*: A resource sharing case study. In *Declarative Agent Languages and Technologies*, volume 2990 of *LNAI*, pages 243–262. Springer-Verlag, May 2004.

[12] M. Alberti, M. Gavanelli, E. Lamma, F. Riguzzi, and S. Storari. Inducing specification of interacting systems and proving their properties: An approach grounded on computational logic. *Intelligenza Artificiale*, In this issue.

[13] M. Baldoni *et al.* Modeling, verifying and reasoning about web services. *Intelligenza Artificiale*, In press.

[14] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8:203–236, 2004.

[15] V. Bryl, P. Mello, M. Montali, P. Torroni, and N. Zannone. B-tropos: Agent-oriented requirements engineering meets computational logic for declarative business process modeling and verification. In *Computational Logic in Multi-Agent Systems VIII*, LNAI. Springer-Verlag, 2008.

[16] M. Bugliesi, E. Lamma, and P. Mello. Modularity in logic programming. *Journal of Logic Programming*, 19/20:43–502, May/Jun 1994. Special Issue on "10 years of Logic Programming.".

[17] F. Chesani, M. Gavanelli, M. Alberti, E. Lamma, P. Mello, and P. Torroni. Specification and verification of agent interaction using abductive reasoning (tutorial paper). In *Computational Logic in Multi-Agent Systems VI*, volume 3900 of *LNAI*, pages 243–264. Springer-Verlag, 2006.

[18] CLIMB: Computational logic for the verification and modeling of business processes and choreographies, 2008. `http://lia.deis.unibo.it/research/climb`.

[19] M. Fisher, R. H. Bordini, B. Hirsch, and P. Torroni. Computational logics and agents: a road map of current technologies and future trends. *Computational Intelligence*, 23(1):61–91, 2007.

[20] MASSiVE: sviluppo e verifica di sistemi multi-agente basati sulla logica. `http://www.di.unito.it/massive`.

[21] P. Mello, P. Torroni, M. Gavanelli, M. Alberti, A. Ciampolini, M. Milano, A. Roli, E. Lamma, F. Riguzzi, and N. Maudet. A logic-based approach to model interaction amongst computees. Technical report, SOCS Consortium, 2003. Deliverable D5. SOCS project web site [22].

[22] Societies Of ComputeeS (SOCS): a computational logic model for the description, analysis and verification of global and open societies of heterogeneous computees. IST-2001-32530, 2002-2005. `http://lia.deis.unibo.it/research/socs`.

[23] F. Toni. Multi-agent systems in computational logic: Challenges and outcomes of the SOCS project. In *Computational Logic in Multi-Agent Systems VI*, volume 3900 of *LNAI*, pages 420–426. Springer-Verlag, 2006.

[24] P. Torroni, M. Gavanelli, and F. Chesani. Argumentation in the semantic web. *IEEE Intelligent Systems*, 22(6):66–74, Nov/Dec 2007.

[25] W. van der Aalst, B. van Dongen, C. Günther, R. Mans, A. A. de Medeiros, A. Rozinat, V. Rubin, M. Song, H. Verbeek, and A. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Application and Theory of Petri Nets and Other Models of Concurrency (ICATPN 2007)*, volume 4546 of *LNCS*, pages 484–494. Springer-Verlag, 2007.

[26] W. M. P. van der Aalst and M. Pesic. Decserflow: Towards a truly declarative service flow language. In M. Bravetti, M. Núñez, and G. Zavattaro, editors, *Web Services and Formal Methods, Third International Workshop, WS-FM 2006 Vienna, Austria, September 8-9, 2006, Proceedings*, volume 4184 of *LNCS*, pages 1–23. Springer, 2006.

Sulla decidibilità di programmi FDNC
On the decidability of FDNC programs

P. A. Bonatti

SOMMARIO/*ABSTRACT*

Questo articolo introduce una nuova dimostrazione della decidibilità del controllo di consistenza per i programmi FDNC sotto la semantica dei modelli stabili, basandosi su *splitting sequences* regolari. Con questa tecnica, riusciamo a rilassare leggermente la definizione di programmi FDNC e muoviamo un primo passo verso l'analisi delle relazioni tra programmi FDNC e la teoria dei programmi finitamente ricorsivi.

We provide a new decidability proof for the consistency of FDNC programs under the stable model semantics, based on regular splitting sequences. With this technique, we can slightly relax the definition of FDNC programs and make a first step towards a precise understanding of the relationships between FDNC programs and finitely recursive programs.

Keywords: Answer set programming, finitely recursive, finitary, and FDNC programs, module sequences.

1 Introduction

Some of the recent works of Alberto concern modal extensions of logic programming [5, 1]. A major motivation for those programs is reasoning about actions and change. In this setting, nonmonotonic constructs such as negation as failure are extremely useful to encode compactly the frame axiom and action consequences. However, for a long time such features could be supported only by forbidding function symbols, in order to ensure decidability.

Later results dropped this restriction. *Finitary programs* [4] preserve decidability even in the presence of infinite domains. This is achieved at the cost of restrictions on the cycles in dependency graphs containing an odd number of negative edges. Such limitations imply restrictions on the constraints (in the form of *denials* like $\leftarrow A_1, \ldots, A_n$, for example) that can be encoded in a finitary program.

FDNC programs [8] adopt a different strategy. They restrict the syntax to (a skolemized form of) 2-variable guarded logic and avoid the restrictions on cycles and constraints.

In this paper we reformulate the decidability of the consistency check for FDNC programs in terms of regular splitting sequences. In this way we slightly generalize a decidability result published in [8].

2 Preliminaries

We assume the reader to be familiar with Logic Programming and the stable model semantics [2]

Disjunctive logic programs are sets of (disjunctive) rules

$$A_1 \vee A_2 \vee ... \vee A_m \leftarrow L_1, ..., L_n \quad (m > 0, n \geq 0),$$

where each A_j ($j = 1, ..., m$) is a logical atom and each L_i ($i = 1, ..., n$) is a *literal*, that is, either a logical atom A or a negated atom $\texttt{not}\ A$.

If r is a rule with the above structure, then let $head(r) = \{A_1, A_2, ..., A_m\}$ and *body*$(r) = \{L_1, ..., L_n\}$. Moreover, let *body*$^+(r)$ (respectively *body*$^-(r)$) be the set of all atoms A s.t. A (respectively $\texttt{not}\ A$) belongs to $body(r)$.

The ground instantiation of a program P is denoted by $Ground(P)$, and the set of atoms occurring in $Ground(P)$ is denoted by $atom(P)$. Similarly, $atom(r)$ denotes the set of atoms occurring in a rule r.

A Herbrand model M of P is a *stable model* of P iff $M \in \mathsf{lm}(P^M)$, where $\mathsf{lm}(X)$ denotes the set of least models of a positive (possibly disjunctive) program X, and P^M is the *Gelfond-Lifschitz transformation* of P, obtained from $Ground(P)$ by (i) removing all rules r such that *body*$^-(r) \cap M \neq \emptyset$, and (ii) removing all negative literals from the body of the remaining rules.

Disjunctive programs may have one, none, or multiple stable models. We say that a program is *consistent* if it has at least one stable model; otherwise the program is *inconsistent*. A *skeptical* consequence of a program P is any formula satisfied by all the stable models of P. A *credulous* consequence of P is any formula satisfied by at least one stable model of P.

The *dependency graph of a program* P is a labelled directed graph, denoted by $DG(P)$, whose vertices are the ground atoms of P's language. Moreover, there exists an edge from A to B iff for some rule $r \in Ground(P)$, $A \in head(r)$ and either B occurs in $body(r)$, or $B \in head(r)$.

An atom A *depends* on B if there is a directed path from A to B in the dependency graph.

A disjunctive program P is *finitely recursive* [4, 3] iff each ground atom A depends on finitely many ground atoms in $DG(P)$.

A *FDNC program* is a set of disjunctive rules conforming to any of the following schemata:

(R1) $A_1(x) \vee ... \vee A_n(x) \leftarrow (\mathtt{not})B_0(x), ..., (\mathtt{not})B_l(x)$
(R2) $R_1(x,y) \vee ... \vee R_n(x,y) \leftarrow$
$(\mathtt{not})P_0(x,y), \ldots, (\mathtt{not})P_l(x,y)$
(R3) $R_1(x, f_1(x)) \vee ... \vee R_n(x, f_n(x)) \leftarrow$
$(\mathtt{not})P_0(x, g_0(x)), \ldots, (\mathtt{not})P_l(x, g_l(x))$
(R4) $A_1(y) \vee ... \vee A_n(y) \leftarrow$
$(\mathtt{not})B_0(Z_0), ..., (\mathtt{not})B_l(Z_l), R(x,y)$
(R5) $A_1(f(x)) \vee ... \vee A_n(f(x)) \leftarrow$
$(\mathtt{not})B_0(W_0), ..., (\mathtt{not})B_l(W_l), R(x, f(x))$
(R6) $R_1(x, f_1(x)) \vee ... \vee R_n(x, f_n(x)) \leftarrow$
$(\mathtt{not})B_0(x), ..., (\mathtt{not})B_l(x)$
(R7) $C_1(\vec{c}_1) \vee ... \vee C_n(\vec{c}_n) \leftarrow (\mathtt{not})D_1(\vec{d}_1), ..., (\mathtt{not})D_l(\vec{d}_l)$

where $n, l \geq 0$, $Z_i \in \{x, y\}$, $W_i \in \{x, f(x)\}$, and each $\vec{c}_i$, $\vec{d}_i$ is a tuple of one or two constants. Each rule r must be *safe*, i.e., each variable must occur in $body^+(r)$. Moreover at least one rule of type (R7) must be a fact.

Our results depend on a *splitting theorem* that allows to construct stable models in stages. In turn, this theorem is based on the notion of *splitting set* of a program P [2],[6], that is, any set U of atoms such that, for all rules $r \in Ground(P)$, if $head(r) \cap U \neq \emptyset$ then $atom(r) \subseteq U$. The set of rules $r \in Ground(P)$ such that $head(r) \cap U \neq \emptyset$ is called the *bottom* of P relative to the splitting set U and is denoted by $bot_U(P)$.

The partially evaluated complement of the bottom program determines the rest of each stable model. The *partial evaluation* of a ground logic program P with splitting set U w.r.t. a set of ground atoms X is the program $e_U(P, X)$ defined as follows:

$$\begin{aligned} e_U(P,X) = \{\, r' \mid\ & \mathit{there\ exists}\ r \in P\ \mathit{s.t.} \\ & (body^+(r) \cap U) \subseteq X,\ (body^-(r) \cap U) \cap X = \emptyset, \\ & head(r') = head(r),\ body^+(r') = body^+(r) \setminus U, \\ & body^-(r') = body^-(r) \setminus U \,\}\,. \end{aligned}$$

We are finally ready to formulate the splitting theorem.

Theorem 1 (Splitting theorem [6]) *Let U be a splitting set for a logic program P. An interpretation M is a stable model of P iff $M = J \cup I$, where*

1. *I is a stable model of $bot_U(P)$, and*
2. *J is a stable model of $e_U(Ground(P) \setminus bot_U(P), I)$.*

The splitting theorem has been extended to *transfinite sequences* in [7]. A (transfinite) sequence is a family whose index set is an initial segment of ordinals, $\{\alpha : \alpha < \mu\}$. The ordinal μ is the *length* of the sequence.

A sequence $\langle U_\alpha \rangle_{\alpha<\mu}$ of sets is *monotone* if $U_\alpha \subseteq U_\beta$ whenever $\alpha < \beta$, and *continuous* if, for each limit ordinal $\alpha < \mu$, $U_\alpha = \bigcup_{\nu<\alpha} U_\nu$. A sequence with $\mu = \omega$ is *smooth* if each of its elements is finite.

Definition 2 [Lifschitz-Turner, [7]] A *splitting sequence* for a program P is a monotone, continuous sequence $\langle U_\alpha \rangle_{\alpha<\mu}$ of splitting sets for P s.t. $\bigcup_{\alpha<\mu} U_\alpha = atom(P)$.

Lifschitz and Turner generalized the splitting theorem to splitting sequences.

Theorem 3 (Splitting sequence theorem [7]) *Let P be a disjunctive program. M is a stable model of P iff there exists a splitting sequence $\langle U_\alpha \rangle_{\alpha<\mu}$ such that*

1. *M_0 is a stable model of $bot_{U_0}(P)$,*
2. *for all successor ordinals $\alpha < \mu$, M_α is a stable model of $e_{U_{\alpha-1}}(bot_{U_\alpha}(P) \setminus bot_{U_{\alpha-1}}(P), \bigcup_{\beta<\alpha} M_\beta)$,*
3. *for all limit ordinals $\lambda < \mu$, $M_\lambda = \emptyset$,*
4. *$M = \bigcup_{\alpha<\mu} U_\alpha$.*

3 Revised decidability results

We first observe that strictly speaking, FDNC programs are not always finitely recursive, due to the presence of *local variables*, i.e. variables that occur in the body of a rule and not in its head. Such variables arise in instances of rule schema (R4); in particular x occurs only in the body. However it is not hard to verify that the following proposition holds:

Proposition 4 *If an atom $R(t,u)$ belongs to a stable model of an FDNC program, then either $u = f(t)$ for some function symbol f, or (t,u) is one of the vectors of constants $\vec{c}_i$ occurring in the head of some instance of (R7).*

It follows that each rule of the form (R4) can be replaced by a finite number of its instances:

- one for each substitution $[y/f(x)]$, where f is a function symbol occurring in the program;
- one for each substitution $[x/a_1, y/a_2]$ for each vector of constants $\vec{c}_i = (a_1, a_2)$ occurring in the head of some instance of schema (R7).

By Proposition 4, such transformation preserves the set of stable models of the given FDNC program. Moreover, the transformation removes all local variables so the transformed program is a finitely recursive FDNC program.

With a similar argument we can further normalize FDNC programs, restricting the set of atoms that may occur in a rule head. Each instance of schema (R2) can be replaced by a finite number of its instances by analogy with the previous case. By Proposition 4, such transformation preserves the set of stable models of the given FDNC program. Moreover, the transformation specializes the heads of the instances of (R2) so that the following lemma holds:

Lemma 5 *Every FDNC program is equivalent to a FDNC program with no rules of the form* (R2) *or* (R4).

Corollary 6 *Every FDNC program P is equivalent to a finitely recursive FDNC program P' such that the binary atoms occurring in $Ground(P')$ are of the form $R(t, f(t))$ (for some function symbol f) or $R(\vec{c}_i)$, where $\vec{c}_i$ occurs in the head of some instance of* (R7).

Note that the above program transformation can be effectively computed. Therefore, from now on, we shall focus without loss of generality on *normal FDNC programs*, that we define as programs whose rules conform to some of the schemata (R1), (R3), (R5), (R6), and (R7).

In the following, let P be a given normal FDNC program, and let us construct a suitable splitting sequence for P. First take any effective enumeration $t_1, t_2, \dots, t_i, \dots$ of the ground compound terms of P's language, such that each term t_i precedes all the terms larger than t_i (in terms of the number of function symbol occurrences). For all such ground terms t_i, we shall denote by $\hat{U}_i$ the set of all ground atoms $A(t_i)$ and $R(t_i, f(t_i))$, for all function symbols f. Now a *canonical splitting sequence* for P can be defined as follows:

- let U_0 be the set of all atoms of the form $A(c)$, $R(c, d)$, or $R(c, f(c))$, where c and d are constants;
- let $U_{i+1} = U_i \cup \hat{U}_{i+1}$.

Since P has no rules conforming to (R2) or (R4), it is easy to check that $\langle U_i \rangle_{i<\omega}$ is indeed a splitting sequence for $Ground(P)$.

Moreover, note that by definition, canonical sequences are smooth, as U_0 and the sets $\hat{U}_i$ are all finite.

Another important property of canonical sequences is that the program slices $P_{i+1} = bot_{U_{i+1}}(P) \setminus bot_{U_i}(P)$ they induce are all isomorphic to each other. By isomorphic, we mean that for all $0 < i < j < \omega$, P_j can be obtained from P_i by uniformly replacing t_i with t_j (in symbols, $P_j = P_i[t_i/t_j]$).

Now consider finite sequences of models $\langle M_i \rangle_{i<k}$ with the following properties:

- M_0 is a stable model of $bot_{U_0}(P)$;
- M_{i+1} is a stable model of $e_{U_i}(bot_{U_{i+1}}(P) \setminus bot_{U_i}(P), M_i)$.

We say such a sequence is *blocked* if $M_k = M_j[t_j/t_k]$ for some j such that $1 < j < k$, that is, M_k can be obtained from M_j by replacing term t_j with t_k.

Lemma 7 *Every blocked model sequence $\langle M_i \rangle_{i<k}$ (induced by a canonical splitting sequence $\langle U_i \rangle_{i<\omega}$ for a normal FDNC program P) can be extended to an infinite sequence $\langle M_i \rangle_{i<\omega}$ satisfying the following properties:*

1. *M_0 is a stable model of $bot_{U_0}(P)$;*
2. *M_{i+1} is a stable model of $e_{U_i}(bot_{U_{i+1}}(P) \setminus bot_{U_i}(P), M_i)$.*

Roughly speaking, the idea simply consists in repeating the subsequence $M_j, \dots, M_{k-1}$ forever, replacing the terms $t_j, \dots, t_{k-1}$ as appropriate.

Proof. Let $\langle M_i \rangle_{i<k}$ be a blocked sequence as described in the lemma's statement. Point 1 follows immediately from the hypothesis, so we focus on point 2. Since $\langle M_i \rangle_{i<k}$ is blocked, there exists $j < k$ such that $M_k = M_j[t_j/t_k]$. For all $i > k$, let $m_i = j + (i - k) \bmod (k - j)$ and $M_i = M_{m_i}[t_{m_i}/t_i]$. Moreover, for all $i \geq 0$ let $P_{i+1} = bot_{U_{i+1}}(P) \setminus bot_{U_i}(P)$. As we already pointed out before this lemma, $P_i = P_{m_i}[t_{m_i}/t_i]$. Now, since both the program slices and the models with indexes i and m_i are subject to the same symbol renaming, we have that

$$e_{U_i}(P_i, M_{i-1}) = e_{U_{m_i}}(P_{m_i}, M_{m_i-1})[t_{m_i}/t_i] \,.$$

Since semantics does not depend on symbol names and by assumption M_{m_i} is a stable model of $e_{U_{m_i}}(P_{m_i}, M_{m_i-1})$ (as m_i lies between j and k), we clearly have that M_i is a stable model of $e_{U_i}(P_i, M_{i-1})$; this proves point 2. ∎

Now proving decidability is relatively easy. We start by characterizing satisfiability in terms of blocked sequences.

Theorem 8 *M is a stable model of a normal FDNC program P iff M is the limit of the extension (in the sense of Lemma 7) of a blocked sequence $\langle M_i \rangle_{i<k}$ (induced by a canonical splitting sequence $\langle U_i \rangle_{i<\omega}$ for P).*

Proof. (*If*) Suppose M is the limit of a sequence $\langle M_i \rangle_{i<\omega}$ such that $\langle M_i \rangle_{i<k}$ is a blocked sequence and such that:

1. M_0 is a stable model of $bot_{U_0}(P)$;
2. M_{i+1} is a stable model of $e_{U_i}(bot_{U_{i+1}}(P) \setminus bot_{U_i}(P), M_i)$.

Note that each program slice $P_{i+1} = bot_{U_{i+1}}(P) \setminus bot_{U_i}(P)$ contains only atoms from $U_{i+1} \setminus U_{i-1}$ (because P is a normal FDNC program). Therefore the partial evaluation of P_{i+1} does not depend on the atoms in U_{i-1}, that is, for all $i < \omega$,

$$e_{U_i}(P_{i+1}, \bigcup_{j \leq i} M_j) = e_{U_i}(P_{i+1}, M_i) \,.$$

Then properties 1 and 2 above entail the properties required by the splitting sequence theorem (for $\mu = \omega$). It follows that the limit $M = \bigcup_{i<\omega} M_i$ is a stable model of P.

(*Only if*) Suppose that M is a stable model of P. Let $M_0 = M \cap U_0$ and for all $i < \omega$, let $M_{i+1} = M \cap (U_{i+1} \setminus U_i)$. By the splitting theorem, M_0 is a stable model of $bot_{U_0}(P)$. Moreover, by applying the splitting theorem twice for all i, we have that each M_{i+1} is a stable model of $e_{U_i}(bot_{U_{i+1}}(P) \setminus bot_{U_i}(P), \bigcup_{j\leq i} M_j)$ that, as we pointed out in the *If* part of the proof, equals $e_{U_i}(bot_{U_{i+1}}(P) \setminus bot_{U_i}(P), M_i)$. Then we are only left to show that the sequence $\langle M_i \rangle_{i<\omega}$ contains a blocked prefix $\langle M_i \rangle_{i<k}$, that is, for some j and k such that $0 < j < k < \omega$, $M_k = M_j[t_j/t_k]$.

To see this, observe that by definition for all $i > 0$, M_i is a subset of $\hat{U}_i$, and $\hat{U}_i$ is isomorphic to $\hat{U}_1$, that is, $\hat{U}_i = \hat{U}_1[t_1/t_i]$ and $|\hat{U}_i| = |\hat{U}_1|$. It follows that for all $i > 0$ there exists $S_i \subseteq \hat{U}_1$ such that $M_i = S_i[t_1/t_i]$. Since $\hat{U}_1$ is finite, there must be two indexes j and k and a set $S \subseteq \hat{U}_1$ such that $0 < j < k \leq 2^{|\hat{U}_1|}$, $M_j = S[t_1/t_j]$, and $M_k = S[t_1/t_k]$. Consequently, $M_k = S[t_1/t_j][t_j/t_k] = M_j[t_j/t_k]$, which completes the proof. ∎

Corollary 9 *A normal FDNC program P has a stable model iff P has a blocked model sequence $\langle M_i \rangle_{i<k}$ (induced by a canonical splitting sequence $\langle U_i \rangle_{i<\omega}$ for P) with $k \leq 2^{|\hat{U}_1|}$.*

Corollary 10 *Deciding whether a FDNC program P is consistent is decidable.*

Proof. Consistency can be nondeterministically checked as follows: First normalize P. Next for $i = 1, \ldots, 2^{|\hat{U}_1|}$, build the program $e_{U_i}(P_{i+1}, M_i)$ and pick up one of its stable models M_{i+1}; if no such model exists, then return false. Check whether M_{i+1} is isomorphic to some previous M_j; if so, return true. Otherwise repeat the loop, or return false if the end of the loop is reached. Clearly, this algorithm returns true in at least one run iff P has a blocked model sequence $\langle M_i \rangle_{i<k}$ with $k \leq 2^{|\hat{U}_1|}$. By the above corollary, it follows that the algorithm solves the consistency problem for P. ∎

Our results do not need all the restrictions placed on FDNC programs. Proposition 4 holds even when the program is not safe, provided that the rules conforming to (R2) have nonempty bodies. The other proofs do not depend on safeness. In this sense, our results are slightly more general than those in [8].

4 Conclusions and future work

We have given an alternative proof of a decidability result of [8] for FDNC programs by proving that a consistent normal FDNC program has always a stable model which is the limit of a regular sequence $\langle M_i \rangle_{i<\omega}$ of stable models of the finite programs $e_{U_i}(P_{i+1}, M_i)$. Such a regular sequence can be finitely represented by a blocked sequence $\langle M_i \rangle_{i<k}$.

The term *blocked* is deliberately inspired by the notion of blocking in tableaux for modal and description logics. The intuitions in all these areas are analogous, and the goals are the same, namely decidable reasoning in the presence of infinite domains through a finite representation of infinite regular models.

We are planning to complete this investigation by characterizing credulous and skeptical reasoning and their computational complexity in terms of blocked model sequences. In particular, in order to provide effective reasoning methods, we are going to exploit the fact that normal FDNC programs are finitely recursive; for such programs the sequence of bottom programs induced by a smooth splitting ω-sequences is consistent iff the entire program is consistent. The consistency of the bottoms can be proved by (a suitable adaptation of) Lemma 7.

It will be interesting to inspect applications of these ideas to modal extensions of logic programming, in the spirit of [5], possibly exploiting the translation in [1].

REFERENCES

[1] M. Baldoni, L. Giordano, and A. Martelli. Translating a modal language with embedded implication into horn clause logic. In *ELP*, volume 1050 of *LNCS*, pages 19–33. Springer, 1996.

[2] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, 2003.

[3] S. Baselice, P.A. Bonatti, and G. Criscuolo. On finitely recursive programs. In *ICLP 2007*, volume 4670 of *LNCS*, pages 89–103. Springer, 2007.

[4] Piero A. Bonatti. Reasoning with infinite stable models. *Artif. Intell.*, 156(1):75–111, 2004.

[5] Laura Giordano, Alberto Martelli, and Camilla Schwind. Ramification and causality in a modal action logic. *J. Log. Comput.*, 10(5):625–662, 2000.

[6] V. Lifschitz and H. Turner. Splitting a Logic Program. In *Proc. of the 12th Int. Conf. on Logic Programming*, MIT Press Series Logic Program, pages 581–595. MIT Press, 1995.

[7] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *International Conference on Logic Programming*, pages 23–37, 1994.

[8] M. Simkus and T. Eiter. FDNC: Decidable nonmonotonic disjunctive logic programs with function symbols. In *LPAR 2007*, volume 4790 of *Lecture Notes in Computer Science*, pages 514–530. Springer, 2007.

Evolving Reactive Logic Programs
Programmi Logici Reattivi Evolutivi

Jose Julio Alferes Federico Banti Antonio Brogi

SOMMARIO/*ABSTRACT*

In questo articolo descriviamo brevemente l'attivitá di ricerca che abbiamo portato avanti negli ultimi anni sui programmi logici dinamici. Dopo aver rivisto i nostri contributi al consolidamento dei fondamenti semantici dei programmi logici dinamici, descriviamo un semplice formalismo —basato su programmi logici dinamici— per ragionare su azioni ed una sua recente estensione che permette di specificare ed eseguire programmi reattivi del tipo evento-condizione-azione.

In this paper we briefly describe the research activity that we have been carrying over during the last years on dynamic logic programs. After reviewing our contributions to strengthening the semantics foundations of dynamic logic programs, we describe a simple formalism to reason about actions —based on dynamic logic programs— and its recent event-condition-action extension that supports the specification and the execution of reactive programs.

Keywords: Logic programs, dynamic knowledge, action description languages, event-condition-action languages.

1 Introduction

Research in Artificial Intelligence (AI) is concerned with producing machines to automate tasks requiring intelligent beaviour. An important problem to face when implementing AI applications is how to represent knowledge, and how to extract information from such knowledge. This area of research is known as knowledge representation (KR) and reasoning. The dominant approach in KR is to define symbolic paradigms based on some form of logic, usually consisting of crude facts and more sophisticated logic formulas. Together, facts and formulas form the knowledge base (KB) of the AI application. Many tasks for AI applications also demand to perform some kind of actions. Hence actions, and possibly the effects of actions, should be representable in the KR framework, and the mechanism specifying when an action must be performed must be defined. Moreover, usually interactive application continually receive external inputs in the form of messages, perceptions, commands and so on. Such inputs can be considered as events to which the AI application is supposed to react in an intelligent way. Reactivity is a key feature in dynamic domains, where changes frequently occur. Among the existing proposals for programming reactive behaviour, *Event-Condition-Action* (ECA) languages distinguish themselves for their flexibility and intuitive syntax and semantics.

Dynamic domains also demand AI applications for taking into account frequent changes and consequently updating their KBs. The required updates surely involve the extensional part of the knowledge base (facts), but occasionally it may be necessary to update also the intentional part (logic formulas) to represent the fact that the very rules of the domain changed. Moreover, for adapting to the new situation, besides knowledge updates, it might be necessary to update the beaviour of the AI applications, i.e. the reactive mechanisms themselves. These updates may be the result of external inputs, but it might be necessary for the application to perform actions leading to self-updates. Moreover, besides what could be called basic actions like, for instance, insertion and deletion of facts and formulas, developera may want to specify more sophisticated actions obtained by combining the basic ones.

Among the existing formalisms for KR, Logic Programming (LP) has a simple logic-based syntax, formal declarative semantics and implemented inference systems. In the past years, part of the research on LP focused on representing dynamic knowledge, i.e. knowledge that is constantly self-updated, leading to the *dynamic logic programming* (DyLP) framework [5, 9, 10, 12, 13]. Taking advantage of the established results in the field, we developed a (dynamic) LP framework for programming AI applications satisfying the above listed features.

In this paper, we first review (Section 2) our contributions to strengthening the semantics foundations of dynamic logic programs that yielded a refined stable-model based semantics and a well-founded semantics for this class of logic programs. We then describe (Section 3) a simple formalism (EAPs) to reason about the effects of actions —based on dynamic logic programs and on the LP update language *Evolp* [4]— and its recent event-condition-action extension *ERA* that supports the specification and the execution of reactive programs. As we will see, *ERA* supports the specification and the execution of reactive programs, by detecting (simple and complex) events, by performing (simple and complex) actions and by allowing self-updates. Since ERA can also encode EAPs, it hence satisfies the features listed earlier in this Introduction. Finally some conclusions and directions for future work are discussed (Section 4).

We assume the reader is familiar with logic programming and the stable models and well-founded semantics and refer to [6] for details on syntax and semantics of LPs.

2 Dynamic Logic Programs

Dynamic Logic Programs represent evolving knowledge. Syntactically, a DyLP $\mathcal{P}$ is a *sequence* $P_1, \ldots, P_n$ (rather than a single program) of generalized logic programs (GLPs), viz., programs where rule heads may be negative literals. P_1 represents the initial knowledge and the other $P_i s$ are supervenient updates representing the evolution of the described situation. Given two updates P_i, P_j, of a DyLP $\mathcal{P}$, P_j is said to be *more recent* than P_i if P_j follows P_i in the sequence $\mathcal{P}$. In the past years, several semantics have been defined for providing a meaning to DyLPs [5, 9, 10, 12, 13]. These semantics are extensions of the stable model semantics of normal logic programs, in the sense that, whenever the considered DyLP is a single normal program P, the models of P in the considered semantics for DyLPs coincide with the stable models of P. Another common denominator of these semantics, is the *causal rejection principle* [10, 12]. This principle states that *a model M of a DyLP $\mathcal{P}$ must fulfill a rule τ in an update of $\mathcal{P}$*, unless *there exists a rule in a more recent update that is in conflict with τ and whose body is true in* M. Two rule τ and η are said to be *in conflict* if they have complementary heads, viz., the head of τ is a literal A and the head of η is $not\, A$ or viceversa. The principle allows a more recent rule to specify an *exception* to an older one, thus allowing to update previous beliefs.

The semantics for DyLPs based on the causal rejection principle coincide on large classes of programs but disagree on some examples and, at the time we started our investigation, there was no general agreement on which should be *the* stable model-like semantics for DyLPs based on the causal rejection principle. Moreover, all the semantics defined before we started our investigation show *counterintuitive beaviour* in some well known example. The simpler examples involve tautological updates that happen to change the semantics of a DyLP, while immunity to tautologies is a property generally required to a semantics.

For instance, the single program DyLP

$$\begin{array}{lll} P_1: & not\, rain. & rain \leftarrow cloudy. \\ & cloudy \leftarrow not\, sun. & sun \leftarrow not\, cloudy. \end{array}$$

has one model $\{not\, rain, sun\}$. If we update P_1 with

$$P_2: \; rain \leftarrow rain.$$

another model $\{rain, not\, sun\}$ is allowed. Somehow, the tautology has generated another model by rejecting the rule $not\, rain$. In general, all the known counterintuitive beaviour occur in DyLPs with cyclic dependencies among literals, somehow leading to the addition of undesired models. although a formal definition of *counterintuitive beaviour* and *undesired model* was missing. Our contribution was:

- to formalise the concept underlying such counterintuitive beaviour and to clarify which should be the right semantics for DyLPs by establishing which properties should be satisfied by such semantics, and
- to define a semantics satisfying these properties, thus avoiding the known counterintuitive beaviour.

To achieve these results we defined the *refined extension principle* [1]. The refined extension principle is a criterium stating when the addition of rules to a program should not add more models to its semantics and it enables to formalize the undesired addition of models. Then, we defined the *refined stable model semantics* (or simply refined semantics) for DyLPs that refines the other stable-like semantics for DyLPs. Formally this was achieved by associating to each DyLP $\mathcal{P} = P_1, \ldots, P_n$, an operator over sets of literals $\Gamma^R_{\mathcal{P}}$ and defining the refined models of $\mathcal{P}$ as the fixpoints of $\Gamma^R_{\mathcal{P}}$. The $\Gamma^R_{\mathcal{P}}$ operator is formally defined as follows:

$$\Gamma^R_{\mathcal{P}}(M) = least\left(\rho(\mathcal{P}) \setminus Rej^R(\mathcal{P}, M) \cup Def(\mathcal{P}, M)\right)$$

where $\rho(\mathcal{P})$ is the multiset of all the rules appearing in any program of the sequence $\mathcal{P}$ and $Rej^R(\mathcal{P}, M)$ is the multiset of all the rules τ in some update P_i of $\mathcal{P}$ for which there exist a rule η in some update P_j with $i \leq j$ such that τ and η are in conflict and the body of η is true in M. Finally $Def(\mathcal{P}, M)$ is the set of default assumptions, i.e. the set of all the negative literals $not\, A$ such that there exists no rule in $\mathcal{P}$ whose head is A and whose body is true in M.

The refined semantics was proved to satisfy the refined extension principle and the causal rejection one. Moreover, we extended the concept of *well supported models* [6] to DyLPs and proved that the refine models of a DyLP are exactly its well supported models.

A further result was the definition of a *well founded semantics for DyLPs* [8]. The well founded semantics is a skeptical approximation of the stable model one. From a practical point of view, the well founded semantics has less expressivity (for instance it does not allow to express logic constraints) and less inference power (it allows to derive less conclusions). On the other hand, the well founded semantics is computationally less expensive than the stable model semantics. Indeed, determining a (refined) stable model of a (dynamic or generalized) logic program is a NP-complete problem, while the computation of the well founded model of a normal logic program has polynomial complexity.

Moreover, unlike the stable model one, the well founded semantics is always defined and, according to it, a program can be queried about specific information without the need to compute its whole semantics. Due to these features, the well founded semantics is a better candidate than the stable model one for applications that are time-committed and require to process huge amount of data, like most of real world database related applications.

We defined a well founded semantics for DyLPs that extends the well founded semantics for normal LPs and approximates the refined one, in the sense that (as for normal LPs) the well founded model of a DyLP is a subset of any of its refined models. Moreover, the well founded semantics for DyLPs preserves the good features shown for the class of normal and generalized LPs, i.e. the well founded model always exists, its computation is polynomial, and a DyLP can be queried about specific information without the need to compute its whole semantics.

The well founded model was defined as the least fixpoint of an operator $\Gamma\Gamma^R$, combining the Γ^R operator used for defining the refined model semantics with another operator Γ used for defining another semantics for DyLPs i.e. the *dynamic stable model semantics* [12].

3 Reasoning about and executing actions

After strngthening the formal foundation of dynamic logic programs, we turned our attention to the the problem of programming self-updatable AI applications capable of reasoning about and executing actions. A bridge between dynamic KR via DyLPs and this kind of applications was already established by the family of *LP updates languages* [4, 10, 12]. These languages are built on the top of a DyLP semantics and, besides representing dynamic and constantly updated knowledge, they allow one to specify how a KB should be updated. Among these formalism the Evolp language [4] has a particularly simple, but highly expressive syntax and semantics, and hence it was chosen as the starting point of our investigation. Evolp is a language for building sequences of DyLPs starting from an original program. Syntactically, Evolp extends the language of LP with new atoms $assert(r)$ where r is a rule. An Evolp programs evolves passing from the current state to the successive one, by updating the program with all the rules r such that the atom $assert(r)$ is true in the current state.

A widely used way to describe and reason about the effects of actions are *action description programs* written in specific formalisms called *action description languages* [11]. We defined an action description language of our own, christened Evolving Action Programs (EAPs) [2]. EAPs are defined as a macro language on top of Evolp in the sense that every statement in EAPs is syntactic notation for a set of Evolp statements and the semantics of an EAP is given by the semantics of the corresponding Evolp program.

Syntactically, an EAP statement can be:

- an inertial declaration $inertial(f)$,
- a static, logic programming-like rule $L \leftarrow L_1, \dots L_n$,
- a dynamic rule **effect(**$H \leftarrow B$**)** $\leftarrow Cond.$

The meaning of an inertial declaration $inertial(f)$, where f is an atom (usually called a *fluent* in the context of action description languages) is that the truth value of f is preserved in time unless it changes as an effect of the execution of an action. A static rule describes the (static) rules of the environment by expressing correlations among fluents. A dynamic rule expresses the effect of the execution of actions. Syntactically, the effect $H \leftarrow B$ is a static rule, while $Cond$ is a conjunction of action literals representing actions being or not executed and fluent literals representing preconditions for the considered effects to take place.

The expressivity of EAPs was compared with that of the action languages $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ (see [11] for a detailed description of these languages) and for each of these languages, a modular embedding of their action programs into EAPs was defined. Moreover, being based on DyLPs, EAPs are shown to be particularly suitable for encoding successive elaborations or updates of an action description problem.

Besides reasoning about the effects of actions, we also needed a formalism for executing them. This was achieved by defining an ECA formalism called ERA (after Evolving Reactive Algebraic programs) [3]. Along with inference logic programming rules, ERA presents two new forms of rules for specifying the execution of actions, i.e. *active* and *inhibition* rules of the form, respectively:

$$\textbf{On } Event \ \textbf{ If } Condition \ \textbf{Do } Action. \quad (1)$$

$$\textbf{When } B \ \textbf{Do } not \ Action. \quad (2)$$

where $Event$ is an *event literal* encoding the occurrence of an event and $Condition$ is a conjunction of literals expressing the condition under which an $Action$ (syntactically an atom) is executed. Finally, B is a conjunction of literals expressing conditions under which $Action$ should *not* be executed. Both events and actions can be basic or complex ones. Complex events and actions are obtained by combining simple ones via an *event* and an *action* algebra.

Events occur at a given instant and are volatile information. Basic events may be external, representing incoming inputs and commands or internal, raised by the system itself. The event algebra allows to combine events occurring simultaneously or at different time points. For instance, the complex event $A(e_1, e_2, e_3)$, where $A/3$ is a ternary operator and the e_is are events, occurs at instant i iff e_3 occurs at instant i, e_1 occurred at some previous instant and e_2 did not occurr in between.

Actions represent operations to be executed. Basic action can be external, representing some external operation to be executed, or internal. As for events, basic actions can be combined by an algebra of operators for specifying flow of operations. For instance, given two action a_1 and a_2, action $a_1 \triangleright a_2$ specifies that action a_2 must be executed after a_1, while action $\|(a_1, a_2)$ specifies that a_1 and a_2 can be concurrently executed.

Among internal actions, particularly important ones are the *assertion* and the *deletion* of facts and rules. While deletion removes facts and rules from the KB, the assertion of rules causes the application to update itself by a new fact, an inference, an active or an inhibition rule. New facts and inference rules are incorporated by the underlying DyLP semantics (that can be the refined as well as the well founded one). Also new active and inhibition rules are incorporated by the underlying DyLP semantics. Assertions of rules of the forms (1) and (2) are translated, respectively, into the LP updates

$$\begin{array}{l} Action \leftarrow Condition, Event. \\ not\, Action \leftarrow B. \end{array}$$

The underlying DyLP framework allows to establish whether the atom $Action$ is derived or not and, in the former case, the corresponding action is executed. In this way the application can update not only its KB but also its beaviour by asserting new active rules and specifying exceptions to existing active rules by asserting inhibition ones. Moreover, it was proved that every Evolp program, and hence every EAP, can be directly encoded into ERA. Thus ERA is a paradigm capable of both executing and reasoning about actions. In [7] ERA is discussed in detail and compared to existing formalisms for programming reactive behaviour. We simply point out here the two main novelties of ERA, i.e. its self evolution capabilities and featured possibility of both programming the execution of actions and reasoning about their effects.

4 Conclusions and future work

In this paper we have tried to briefly describe the research activity that we have been carrying over during the last years on dynamic logic programs. After reviewing our contributions to strengthening the semantics foundations of dynamic logic programs, we have presented the EAPs formalism to reason about actions, and its recent event-condition-action extension *ERA* that supports the specification and the execution of reactive programs. While space limitations only allowed us to provide an extended abstract of this research activity, more details can be found in the papers [1, 2, 3, 8] and a complete presentation of all the results is reported in [7].

There are several open windows for future work. One of them is the definition of *action query languages* [11], that is, languages for extracting information about the possible evolution of the situations described by EAPs and to address planning issues, e.g., how to determine, given a current state and a goal, a sequence of actions leading to a state satisfying that goal. Another direction for future work are *transactions*. Although the action algebra of ERA allows one to program complex actions, it is still less than adequate for defining transactions. In order to define and execute transactions, the action algebra of ERA should be extended for coping with the execution of ACID transactions as well as of compensation activities.

REFERENCES

[1] J. J. Alferes, F. Banti, A. Brogi, and J. A. Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1), 2005.

[2] J.J. Alferes, F. Banti, A. Brogi. From logic programs updates to action description updates. In J. Leite, P. Torroni (eds.), *CLIMA V*, LNAI, pages 52–77, 2005.

[3] J.J. Alferes, F. Banti, A. Brogi. An event-condition-action logic programming language. *JELIA 2006*, LNAI, pages 29-42, 2006.

[4] J. J. Alferes, A. Brogi, J. A. Leite, L. M. Pereira. Evolving logic programs. *JELIA'02*, LNAI, 2002.

[5] J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming*, 45(1–3):43–70, 2000.

[6] K. R. Apt and R. N. Bol. Logic programming and negation: A survey. *The Journal of Logic Programming*, 19 & 20:9–72, May 1994.

[7] F. Banti. *Evolving Reactive Logic Programs*. PhD thesis, Universitade Nova de Lisboa, 2008.

[8] F. Banti, J.J. Alferes, A. Brogi. Well founded semantics for logic program updates. *IBERAMIA'04*, LNCS 3314, pages 397–407, 2004.

[9] F. Buccafurri, W. Faber, and N. Leone. Disjunctive logic programs with inheritance. *ICLP'99*, 1999.

[10] T. Eiter et al.. A framework for declarative update specifications in logic programs. In *IJCAI*, 2001.

[11] M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on AI*, 16, 1998.

[12] J. A. Leite. *Evolving Knowledge Bases*. Frontiers in Artificial Intelligence and Applications, vol. 81, 2003.

[13] J. A. Leite and L. M. Pereira. Generalizing updates: from models to programs. *LPKR'97*, 1997.

Una fruttuosa esperienza in Logica Computazionale
A valuable experience in Computational Logic

Annalisa Bossi, Nicoletta Cocco
Dipartimento di Informatica, Università Ca' Foscari di Venezia,
via Torino 155, 30172, Venezia, Italy
email: {bossi, cocco}@dsi.unive.it

SOMMARIO/*ABSTRACT*

Illustriamo qui brevemente la nostra esperienza nel campo della verifica e delle trasformazioni dei programmi logici. Pur occupandoci ora di tematiche completamente diverse, verifica di proprietà di sicurezza da un lato e analisi di sistemi biologici dall'altro, continuiamo ad utilizzare proficuamente la nostra precedente esperienza.

In this paper, we briefly describe our esperience in the field of verification and transformation of logic programming. Though now we are working in a completely different field, verification of security properties on one hand and biosystems analysis on the other, our previous experience continues to be a valuable guide.

Keywords: logic programming, termination verification, program transformation

1 Introduction

It is very pleasant to remember the time we spent in working on Logic Programming. The friendship and warmth of the people we met, the enthusiasm and interest in research, the curiosity and joy of young researchers, have been strong reasons for working in this field and to be happy with it.

Our main interest, since the beginning, was analysis and verification of logic programs and program transformation. After more than fifteen years of happy and satisfactory research, we felt the need to enlarge our research field and we tried to export our expertise in Computational Logic to different research topics.

In the following section we briefly resume our main results in the field of logic programming and then we give a brief account of our present research interests.

2 Our contribution to Logic Programming

Programming methodology imposes to focus first on the correctness of a program and only later on its efficiency. This is necessary also in logic programming and it requires both program verification and program optimization tools. Our research in LP has been motivated by these needs, dealing mainly with *transformation systems* and with *analysis techniques*.

Analysis techniques.

Logic programs are declarative in essence, and this is a great advantage for programs prototyping and development. Nevertheless, there are properties which are not directly expressed by the program itself and have to be proved. We proposed a technique for verifying correctness and completeness of a logic program with respect to a Pre/Post declarative specification of data properties [1]. This can be used to guarantee both the correspondence of the program to its intended meaning and the applicability of program transformations. We considered also the operational property on having successes, or finite failures, which is relevant for query correctness and efficiency [5, 6]. Besides, the property of not having finite failures can be used to simplify applicability conditions of program transformation operations.

Techniques for verifying termination.

Termination is an essential property of programs. We considered the problem of verifying *universal termination* of logic programs. This is a rather strong requirement for a query, namely to have only finite LD-derivations[1]. All the methods to solve this problem, if effective, can only provide sufficient criteria for termination. In our works we developed various methods for the analysis of universal termination by considering different classes of programs which can be verified.

We introduced a class of functions to weight the terms occurring in a program (*semilinear norms*) [16, 18]. The norms in this class provide a syntactical characterization

[1]SLD-derivations build with the leftmost selection rule.

of rigid terms, i.e. terms whose weight does not change under substitution. The notion of rigid term generalizes the notion of ground term. We defined a proof method for universal termination, based on Pre/Post conditions which deal with the rigidity of terms and can be derived by the mode and type properties of atoms. In [17] we generalized our previous work by considering also terms with a specified structure by means of *typed norms*. Besides, we studied how mode and type information can be used for characterizing termination properties. We defined the class of *well-moded programs* [31], namely programs which are inductively ”well-formed” with respect to a specified input-output functionality. This allowed us to define and characterize *well-terminating programs*, namely programs for which all well-moded queries have only finite LD-derivations. We proposed also a termination property for general logic programs (programs with negation) [19]. A general program is *typed-terminating* if it terminates for any well-typed query. These definitions lead to sufficient conditions for termination which are compositional and simple to verify.

In [14] we completed our work on the verification of termination properties, by proposing a modular proof technique applicable to hierarchical general programs. Besides, by using mode or type information, it is possible to verify termination incrementally.

Trasformations on logic and Prolog programs.
Program transformations are applied both in program synthesis and in program optimization. For logic programs the “logic” component makes transformations very natural and easy to be studied formally. But, when we move to Prolog programs, non-declarative properties, like termination, cannot be ignored.

At first we focused on *program specialization*, which consists in restricting the applicability of the original program while optimizing it: the specialized program deals with fewer cases but in a more efficient way. Some parts of the computation become redundant, other parts can be pre-computed (partial evaluation). Specialization seems to fit very well logic programs in order to pass from a relational definition to some specific functionalities. We proposed a methodology for specializing a logic program [7] and studied a set of basic transformation operations which allow one (i) to associate a new application domain to the query by means of constraints, and (ii) to propagate them through the program for optimizing it. The set of basic operations includes:

- *new definition*, it defines a new predicate in terms of other predicates already available in the program;
- *unfold*, it substitutes an atom in a clause body with all its definitions;
- *fold*, it substitutes a set of atoms in a clause body with an equivalent atom;
- *prune*, it removes a redundant clause from the program;
- *thin*, it removes a redundant literal from a clause body;
- *fatten*, it adds further literals in a clause body whenever this allows for simplifications;
- *replace*, it substitutes a set of literals in a clause body for another set of literals; it is a generalization of the *thin* and *fatten* operations.

Each operation must produce a program which is equivalent to the original one, but more efficient. Program equivalence depends on the semantics we consider. Hence, we studied these transformation operations with respect to different program semantics. Our effort has been to determine sufficient conditions, simple to verify, for the various operations and semantics. We considered the classic semantics given by the minimal Herbrand model [7] and the semantics given by computed answers substitutions [2]. Moreover, in [8, 9] we considered general programs (with negation) and some semantics for them, such as Fitting’s semantics, Kunen’s semantics, and the Well-founded semantics.

Besides basic transformation operations, we defined *simultaneous replacement* and we studies it with respect to the three-valued completion of a logic program [11].

Any transformation system is a source-to-source rewriting methodology devised to improve the efficiency of a program. Any such transformation should preserve the main properties of the initial program. The transformation operations defined for logic programs do not consider operational properties, among them, termination. These properties become relevant for Prolog programs. To deal with that we followed two approaches.

On one hand, we considered *acyclic programs*, namely programs which terminate for each ground query and any selection rule, and *acceptable programs*, namely programs which terminate for each ground query and leftmost selection rule. For both of them we identified the subclasses of programs closed under unfold and fold operations [20, 11].

In order to be applicable most of the transformations require to reorder the atoms in clause bodies, then in [12] we extended the previous work by considering also a *switch* operation which allows one to reorder consecutive atoms.

On the other hand, in [3, 4] we followed a more operational approach and we defined a *non-increasing* property for a transformation. It is a very strong property which guarantees that the transformation is both preserving universal termination and optimizing, since it cannot increase the depth of the derivation tree associated to a query.

In [13] we considered and analyzed the main systems for transforming logic and Prolog programs. In particular we discuss if they preserve non-declarative properties of the original program and specifically termination properties.

Semantics for logic programs.
Our work on the semantics of logic programming is ruled by the convincement that a semantics should help in understanding the meaning of programs by providing useful notions of observable program equivalences. The *s-semantic approach* (see [26]) provides a methodology to define semantics which enjoy this property. Each semantics in the approach captures some observable properties

of logic programs and allows us to detect when two programs are undistinguishable by observing their behaviors, thus providing a suitable base for program analysis and transformation. Following this approach, we defined the Ω- *semantics*, a compositional semantics for positive logic programs. It provides a refined notion of observational equivalence which takes into account both computed answers and program composition by union of clauses [27].

Most logic programming languages offer some kind of *dynamic scheduling* to increase the expressive power and to control execution. But the presence of dynamic scheduling makes more complex the programs behaviour and more difficult the description of the semantics. *Input consuming* derivations have been introduced and studied in [21, 22, 23] to describe dynamic scheduling while abstracting from the technical details. In [15] we reviewed and compared the different proposals given for dynamic scheduling and in particular for the denotational semantics of programs with input consuming derivations. We also show how they can be applied to termination analysis.

3 Present Research

Verification of security properties.
In the recent years, security has gained more and more importance. In this field, our research focus on *information flow* properties, i.e., security properties that allow one to express constraints on how information should flow among different groups of entities. An interesting feature of these kind of properties, is that they protect the system even against internal attacks performed by, e.g., viruses or Trojan horses.

We study different classes of security properties and conditions to ensure global properties by means of local *unwinding conditions* [25]. Locality allows us to define a proof system which provides a very efficient technique for the development and verification of secure processes [24].

For many practical applications the requirement of a complete absence of any information flow could be stronger than necessary when some knowledge about the environment (context) in which the process is going to run is available. To relax this requirement we introduce a general notion of *secure contexts* for a process [28]. In [29] we moved from a process algebra setting to a standard programming environment. We present a general unwinding framework for the definition of information flow security properties of concurrent programs, described in a standard imperative language, admitting parallel executions on a shared memory.
Biosystems analysis.
Computational biology is a recent field combining computer science and molecular biology to study living beings. We focus our attention on two areas, pattern discovery and system biology.
Pattern discovery. Many biological problems require to blindly search into DNA or protein sequences for relevant signals. Often we may assume that strings which appears ”strangely often” or ”strangely rarely” in such sequences have an associated functional purpose. We studied the techniques for finding such signals and for giving them a compact representation as patterns. In particular we define *maximal patterns* [30], which correspond to the largest subsets of strings which can be grouped together. The set of maximal patterns is unique and very readable, intuitively it represents all possible ”most abstract views” of the strings we are interested in. We propose two different algorithms for computing the set of maximal patterns.
Systems biology is a rather new field studying complex interactions in biological systems. The aim is to model such systems, to formally analyze their properties and to simulate their behaviour. This would make possible to do *in silico* experiments instead of *in vivo* experiments, which may be difficult, or even impossible, to perform on biological systems. Computational logic and formal techniques to specify and analyze concurrent processes can be applied to this field.

4 Acknowledgements

Our thanks to Matteo Baldoni and Cristina Baroglio for organizing this collection in honour of Alberto Martelli, a dear friend and one of the most attracting personality for the Italian Computational Logic community.

REFERENCES

[1] A. Bossi and N. Cocco. Verifying Correctness of Logic Programs. In J. Diaz and F. Orejas, editors, *Proceedings TAPSOFT ’89*, Barcelona, Spain, LNCS 352, pp. 96–110, Springer-Verlag, 1989.

[2] A. Bossi, e N. Cocco. Basic Transformation Operations which preserve Computed Answer Substitutions of Logic Programs. *Journal of Logic Programming,* 16:47–87, 1993.

[3] A. Bossi and N. Cocco. Preserving universal termination through unfold/fold. In G. Levi and M. Rodríguez-Artalejo, editors, *Proceedings ALP’94*, LNCS 850, pp. 269–286, Springer-Verlag, 1994.

[4] A. Bossi and N. Cocco. Replacement Can Preserve Termination. In J. Gallagher, editor, *Proceedings LOPSTR’96*, LNCS 1207, pp.104–129, Springer-Verlag, 1997.

[5] A. Bossi and N. Cocco. Programs without Failures. In N. Fuchs, editor, *Proceedings LOPSTR’97*, LNCS 1463, pp. 28–48, Springer-Verlag,1998.

[6] A. Bossi and N. Cocco. Successes in Logic Programs. In P. Flener, editor, *Proceedings LOPSTR’98*, LNCS 1559, pp. 219–239, Springer-Verlag, 1999.

[7] A. Bossi, N. Cocco, e S. Dulli. A Method for Specializing Logic Programs. *ACM Transactions on Programming Languages and Systems*, 12(2):253–302, 1990.

[8] A. Bossi, N. Cocco, e S. Etalle. On Safe Folding. In M. Bruynooghe and M. Wirsing, editors, *Proceedings PLILP'92*, Leuven, Belgium, LNCS 631, pp. 172–186, Springer-Verlag, 1992.

[9] A. Bossi, N. Cocco, e S. Etalle. Transforming Normal Programs by Replacement. In A. Pettorossi, editor, *Proceedings META'92*, Uppsala, Sweden, LNCS 649, pp. 265–279, Springer-Verlag, 1992.

[10] A. Bossi, N. Cocco, and S. Etalle. Transformation of Left Terminating Programs: the Reordering Problem. In M. Proietti, editor, *Proceedings LOPSTR'95*, LNCS 1048, pp. 33–45, Springer-Verlag, 1995.

[11] A. Bossi, N. Cocco, e S. Etalle. Simultaneous Replacement in Normal Programs. *Journal of Logic and Computation*, 6(1):79–120, 1996.

[12] A. Bossi, N. Cocco, e S. Etalle. Transformation of Left Terminating Programs. In A. Bossi editor, *Proceedings of LOPSTR'99*, Venezia, Italy, LNCS 1817, pp. 156-175, Springer-Verlag, 2000.

[13] A. Bossi, N. Cocco e S. Etalle. Transformation Systems and Nondeclarative Properties. In A. Kakas and F. Sadri editors, *Computational Logic: Logic Programming and Beyond (Essays in honour of Robert A. Kowalski)*. LNAI 2407, pp. 162-186, Springer-Verlag, 2002.

[14] A. Bossi, S. Etalle N. Cocco, and S. Rossi. On Modular Termination Proofs of General Logic Programs. *Theory and Practice of Logic Programming*, 2(3):263–291, 2002.

[15] A. Bossi, S. Etalle N. Cocco, and S. Rossi. Declarative Semantics of Input-Consuming Logic Programs. In M. Bruynooghe, K. Lau editors, *Program Development in Computational Logic - A Decade of Research Advances in Logic-Based Program Development*. LNCS 3049, Springer-Verlag, 2004.

[16] A. Bossi, N. Cocco, and M. Fabris. Proving termination of logic programs by exploiting term properties. In S. Abramsky and T.S.E. Maibaum, editors, *Proceedings CCPSD-TAPSOFT'91*, LNCS 494, pp. 153–180, Springer-Verlag, 1991.

[17] A. Bossi, N. Cocco, and M. Fabris. Typed Norms. In Krieg-Bruckner, editor, *Proceedings ESOP'92*, Rennes, France, LNCS 582, pp. 73–92. Springer-Verlag, 1992.

[18] A. Bossi, N. Cocco, and M. Fabris. Norms on terms and their use in proving universal termination of a logic program. *Theoretical Computer Science*, 124:297–328, 1994.

[19] A. Bossi, N. Cocco, and S. Rossi. Termination of Well-Typed Logic Programs. In H. Sondergaard editor, *Proceedings PPDP'01*, Firenze, Italy, pp.73-81, ACM Press, 2001.

[20] A. Bossi and S. Etalle. Transforming Acyclic Programs. *ACM Transactions on Programming Languages and Systems*, 16(4):1081–1096, July 1994.

[21] A. Bossi, S. Etalle, and S. Rossi. Semantics of well-moded input-consuming logic programs. *Computer Languages*, 26(1):1–25, 2000.

[22] A. Bossi, S. Etalle, and S. Rossi. Properties of input-consuming derivations. *Theory and Practice of Logic Programming*, 2(2):125–154, 2002.

[23] A. Bossi, S. Etalle, S. Rossi, and J.-G. Smaus. Termination of simply-moded logic programs with dynamic scheduling. *ACM Transactions on Computational Logic (TOCL)*, 5(3):470–507, 2004.

[24] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. A proof system for information flow security. M. Leuschel, editor, *Proceedings LOPSTR'02*, LNCS 2664, pp. 2199–218, Springer-Verlag, 2003.

[25] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Verifying persistent security properties. *Computer Languages, Systems and Structures*, 30(3-4):231–258, 2004.

[26] A. Bossi, M. Gabrielli, G. Levi, and M. Martelli. The S-semantics approach: Theory and applications. *The Journal of Logic Programming*, 19 & 20:149–198, May 1994.

[27] A. Bossi, M. Gabbrielli, G. Levi, and M. C. Meo. A compositional semantics for logic programs. *Theoretical Computer Science*, 122(1-2):3–47, 1994.

[28] A. Bossi, D. Macedonio, C. Piazza, and S. Rossi. Information flow in secure contexts. *Journal of Computer Security*, 13(3):391–422, 2005.

[29] A. Bossi, C. Piazza, and S. Rossi. Compositional information flow security for concurrent programs. *Journal of Computer Security*, 15(3):373–413, 2007.

[30] N. Cocco and M. Simeoni, Maximal abstraction of strings. *Dipartimento di Informatica, Università Ca' Foscari di Venezia, Rapporto di ricerca* CS-2007-2, 2007.

[31] S. Etalle, A. Bossi, and N. Cocco. Termination of well-moded programs. *Journal of Logic Programming*, 38(2):243–257, 1999.

La Logica Computazionale in sistemi basati su agenti *Computational Logic in Agent Based Systems*

Paolo Mancarella e Francesca Toni

SOMMARIO/*ABSTRACT*

Viene descritto l'utilizzo della logica computazionale a supporto della formalizzazione ed implementazione di agenti in sistemi multi-agente. In questo ambito è necessario l'uso di varie forme di logica computazionale, tra le quali abduzione, argomentazione e sistemi basati su preferenze. Viene presentato a grandi linee il modello per agenti denominato KGP, nonché una sua estensione in corso di definizione per la modellazione di agenti in ambienti distribuiti quali il Grid e più in generale architetture *service-oriented*

We describe recent work on the deployment of computational logic to support the formalisation and implementation of agents in multi-agent systems. Several forms of computational logic systems are needed in this setting, including abductive, argumentative and preference-based systems. We briefly sketch the agent model called KGP, and an ongoing extension of it which is needed to model agents in distributed settings such as the Grid and, more generally, Service-oriented architectures.

Keywords: Logica computazionale, sistemi multi-agente, abduzione, argomentazione

1 Introduction

Computational Logic (CL) has been successfully adopted, in recent years, in modelling agents within agent based systems. The adoption of CL techniques has the advantage that formal specifications come along with their computational counterparts in the form of provably correct and executable proof procedures. The formal and computational models needed in the agent based settings require an integrated treatment of different features, which can be handled within various extensions of the basic logic programming framework, including abduction, argumentation and constraint logic programming. In this short paper we briefly summarize one such approach which has lead to the defnition of the KGP model for agency, and which is being further developed to cope with the specification of agents in service-oriented applications.

2 The KGP model

KGP is an acronym for **K**nowledge, **G**oals and **P**lan. The model is intended to provide a modular and hierarchical specification of agents equipped with a variety of advanced reasoning features to allow intelligent decision making and behaviour. KGP agents are particularly suited to open dynamic environments where they have to adapt to changes in their environment and they have to function in circumstances where they have incomplete information. Here we give an overview of the KGP agent model and its components [4, 3]. The model relies upon
– an internal (or mental) *state*, holding the agent *K*nowledge base (beliefs), *G*oals (desires) and *P*lans (intentions),
– a set of *reasoning capabilities*,
– a set of *physical capabilities*,
– a set of *transition rules*, defining how the state of the agent changes, and defined in terms of the above capabilities,
– a set of *selection operators*, to enable and provide appropriate inputs to the transitions,
– a *cycle theory*, providing the control for deciding which transitions should be applied when, and defined using the selection operators. The model is defined in a modular fashion, in that different activities are encapsulated within different capabilities and transitions, and the control is a separate module. The model also has a hierarchical structure, depicted in figure 1.

Internal state. This is a tuple $\langle KB_0, \mathcal{F}, \mathcal{C}, \Sigma \rangle$, where:

- KB_0 holds the (dynamic) beliefs of the agent about the external world in which it is situated, as well as a record of the actions it has already executed.

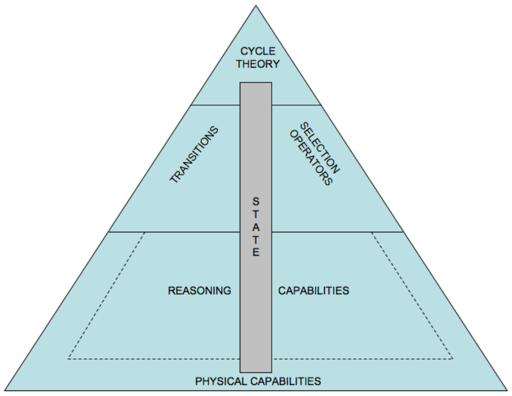


Figure 1: A graphical overview of the KGP model

- $\mathcal{F}$ is a forest of trees whose nodes are *goals*, which may be executable or not. Each tree in the forest gives a hierachical presentation of goals, in that the tree represents the construction of a plan for the root of the tree. The set of leaves of any tree in $\mathcal{F}$ forms a currently chosen plan for achieving the root of the tree. *Executable goals* are *actions* which may be *physical*, *communicative*, or *sensing*. *Non-executable goals* may be *mental* or *sensing*. Only non-executable mental goals may have children, forming (partial) plans for them. Actions have no children in any trees in $\mathcal{F}$. The roots of trees in $\mathcal{F}$ are referred to as *top-level goals*, the executable goals are referred to as *actions*, and non-executable goals which are not top-level goals are referred to as *sub-goals*. Top-level goals are classified as *reactive* or *non-reactive*. Roughly speaking, reactive goals are generated in response to observations, e.g. communications received from other agents and changes in the environment, for example to repair plans that have already been generated. Non-reactive goals, on the other hand, are the chosen desires of the agent. Note that some top-level (reactive) goals may be actions.

- $\mathcal{C}$ is the Temporal Constraint Store, namely a set of constraint atoms in some given underlying constraint language. These basically constrain the time variables of the goals in $\mathcal{F}$. For example, they may specify a time window over which the time of an action can be instantiated, at execution time.

- Σ is a set of equalities instantiating time variables with time constants. For example, when the time variables of actions are instantiated at action execution time, records of the instantiations are kept in Σ.

Reasoning capabilities. These are:

- *Planning*, which generates plans for mental goals given as input. These plans consist of temporally constrained sub-goals and actions designed for achieving the input goals.

- *Reactivity*, which is used to provide new *reactive* top-level goals, as a reaction to perceived changes in the environment and the current plans held by the agent.

- *Goal Decision*, which is used to revise the *non-reactive* top-level goals, adapting the agent's state to changes in its own preferences and in the environment.

- *Identification of Preconditions* and *Identification of Effects* for actions, which are used to determine appropriate sensing actions for checking whether actions may be safely executed (if their preconditions are known to hold) and whether recently executed actions have been successful (by checking that some of their known effects hold).

- *Temporal Reasoning*, which allows the agent to reason about the evolving environment, and to make predictions about properties, including non-executable goals, holding in the environment, based on the (partial) information the agent acquires over its life-time.

- *Constraint Solving*, which allows the agent to reason about the satisfiability of the temporal constraints in $\mathcal{C}$ and Σ.

In the concrete realisation of the KGP model, we have chosen to realise the above capabilities in various extensions of the logic programming paradigm. In particular, we use (conventional) logic programming for Identification of Preconditions and Effects, abductive logic programming with constraints for Planning, Reactivity and Temporal Reasoning, and logic programming with priorities for Goal Decision.

Physical capabilities. In addition to the reasoning capabilities, the agent is equipped with "physical" capabilities, linking the agent to its environment, consisting of

- A *Sensing* capability, allowing the agent to observe that properties hold or do not hold, and that other agents have executed actions.

- An *Actuating* capability, for executing (physical and communicative) actions.

Transitions. The state $\langle KB_0, \mathcal{F}, \mathcal{C}, \Sigma \rangle$ of an agent evolves by applying transition rules, which employ the capabilities as follows:

- *Goal Introduction (GI)*, possibly changing the top-level goals in $\mathcal{F}$, and using Goal Decision.

- *Plan Introduction (PI)*, possibly changing $\mathcal{F}$ and $\mathcal{C}$ and using Planning.

- *Reactivity (RE)*, possibly changing the reactive top-level goals in $\mathcal{F}$ and possibly $\mathcal{C}$, and using the Reactivity capability.
- *Sensing Introduction (SI)*, possibly introducing new sensing actions in $\mathcal{F}$ for checking the preconditions of actions already in $\mathcal{F}$.
- *Passive Observation Introduction (POI)*, changing KB_0 by recording unsolicited information coming from the environment, and using Sensing.
- *Active Observation Introduction (AOI)*, possibly changing Σ and KB_0, by recording the outcome of (actively sought) sensing actions, and using Sensing.
- *Action Execution (AE)*, executing all types of actions and as a consequence changing KB_0 and Σ, and using Actuating.
- *State Revision (SR)*, possibly revising $\mathcal{F}$, and using Temporal Reasoning and Constraint Solving.

Cycle and Selection operators. The behaviour of an agent is given by the application of transitions in sequences, repeatedly changing the state of the agent. These sequences are not determined by fixed cycles of behaviour, as in conventional agent architectures, but rather by reasoning with *cycle theories*. Cycle theories define preference policies over the order of application of transitions, which may depend on the environment and the internal state of an agent. They rely upon the use of *selection operators* for detecting which transitions are enabled and what their inputs should be, as follows:
$-$*action selection* for inputs to AE;
this selection operator uses the Temporal Reasoning and Constraint Solving capabilities; $-$ *goal selection* for inputs to PI;
this selection operator uses the Temporal Reasoning and Constraint Solving capabilities; $-$ *effect selection* for inputs to AOI; this selection operator uses the Identification of Effect reasoning capability;
$-$ *precondition selection* for inputs to SI; this selection operator uses the Identification of Preconditions, Temporal Reasoning and Constraint Solving capabilities.
The provision of a declarative control for agents in the form of cycle theories is a highly novel feature of the model, which could, in principle, be imported into other agent systems. In the concrete realisation of the KGP model, we have chosen to realise cycle theories in the same framework of logic programming with priorities and constraints that we also use for Goal Decision.

2.1 Computational model

One central distinguishing feature of the KGP model, in comparison with other models for agency, including those based on logic programming, is its modular integration within a single framework of abductive logic programming, temporal reasoning, constraint logic programming, and preference reasoning based on argumentation in order to support a diverse collection of capabilities. Each one of these is specified declaratively and equipped with its own provably correct computational counterpart. These computational models are heavily based upon proof procedures for (various extensions of) logic programming. In particular, the operational model for KGP agents relies upon CIFF [2], a proof procedure for abductive logic programming with constraints, and Gorgias [1], for logic programming with priorities. These procedures have been obtained by adapting and suitably extending two existing proof procedures for logic programming, namely Fung and Kowalskis IFF procedure for abductive logic programming for CIFF, and Kakas and Tonis argumentation-based procedure for negation as failure in logic programming for Gorgias. The overall operational models are sound and (in some cases) complete with respect to the abstract KGP model, and form a solid bridge between the KGP model and its implementation within the PROSOCS platform, a prototype implementation using Sicstus Prolog and JXTA [5].

3 Argumentative agents in ARGUGRID

The use of agent technology offers a solution to dynamic service composition in distributed settings such as the Grid and more generally Service-Oriented Architectures. Different services can be associated with autonomous agents that can identify and negotiate, on behalf of service requestors and providers, implementation plans that take into account the requirements of both sides. The ARGUGRID project[1] aims at defining and deploying argumentation-based agents to support the selection and composition of services over the Grid and Service-Oriented Architectures [6]. We have proposed in [7] an agent architecture integrating a number of argumentative modules (for various forms of decision-making), a module for interaction with other agents, "physical" modules for carrying out this interaction via communication, and several data structures. This type of argumentative agents can be seen as a variant of KGP agents. This variant relies upon the use of an argumentation decision-making tool supporting all reasoning capabilities, and it makes use of argumentative protocols for persuasion in negotiation. Argumentative agents are equipped with a specialised internal state, consisting of requirements, abstract or partially instantiated workflows, concrete workflows, planned communicative actions and actions executed in the past by the agent or by others, and arguments. The KGP modular architecture allows us to adopt a specialised set of reasoning capabilities supporting the various forms of decision-making needed in ARGUGRID and inter-agent interaction, as well as a capability for revising the knowledge/beliefs of agents, which is actu-

[1]www.argugrid.eu

ally missing in the original KGP model. Specialised physical capabilities are also needed in this setting to provide suitable forms of inter-agent communications, and finally appropriate transitions encapsulate the new capabilities. In this setting, agents need to be able to perform communicative actions (for requesting services, accepting or refusing the provision of services, etc.) and actions for consulting registries, inquiring about services and their providers. In their internal state, agents store (a selection of) all communicative acts they have participated in, as either speakers or receivers, as well as the set of their current commitments, namely the contracts they have committed to. Basically, argumentative KGP agent are characterised by

- a (transient) state, consisting of
  − a knowledge base, called KB_0 as for the KGP model, but holding communicative acts by or to the agent, acts for consulting registries by the agent, as well as contracts
  − a set of goals, namely requirements by the user "owning" the agent
  − a set of *decisions*, of different kinds (to get services of known types from some yet-to-be-decided providers or from some known providers, or a decision to utter something, or a decision to consult some registry)
  − a set of arguments, providing justifications and reasons for goals and decisions in the state
- a number of extended reasoning capabilities, namely abstract decision-making, social decision-making, communicative reactivity, registry consultation; each capability is supported by an appropriate argumentation system (base)
- a revision capability, for modifying the argumentation systems supporting the various reasoning capabilities
- physical capabilities, namely listening, talking, and consulting
- a set of transitions, namely ADM (using the abstract decision-making capability), SDM (using the social decision-making capability), CR (using the communicative reactivity capability), RC (using the registry consultation capability), R (using the revision capability), LI, TA, CON (using the listening, talking and consulting capabilities, respectively)
- a control, in the form of a conditional policy, that, for each transition, gives one or more possible next transitions depending on whether a number of conditions hold or not.

Here, the consulting capability is intended for accessing information in registries. The reasoning capabilities correspond to the IDM (individual decision-making), SDM (social decision making), and SI (social interaction) modules in [7]. The listening and talking capabilities are special cases of sensing and actuating in the KGP model.

4 Conclusions

We have briefly described work carried out in recent years and still ongoing, which aims at adopting computational logic for the description of agents in agent based systems. The use of computational logic allows us, on one hand to partially fill the gap between agent models and their computational realization. Indeed, the specification of KGP agent is a sort of *executable* specification due to the fact that the computational logic tools adopted in this setting are equipped with suitable concrete proof procedures. On teh other hand, the modularity of the KGP model allows one to extend it naturally to support new forms of reasoning, such as the ones needed in order to model the type of agents needed in service-oriented applications. Again, computational logic tools, based on various forms of argumentation, can be adopted in these settings to support the new type of capabilities needed, such as decision making and negotiation. This is still ongoing work we are carrying out within the ARGUGRID project.

REFERENCES

[1] N. Demetriou and A. C. Kakas. Argumentation with abduction. In *Proceedings of the fourth Panhellenic Symposium on Logic*, 2003.

[2] U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The CIFF proof procedure for abductive logic programming with constraints. *Lecture Notes in Artificial Intelligence*, 3229:680–684, 2004.

[3] A. C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. Declarative agent control. *Lecture Notes in Artificial Intelligence*, 3487:96–110, 2005.

[4] A.C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In R. Lopez de Mantaras and L. Saitta, editors, *Proceedings of the Sixteenth European Conference on Artificial Intelligence, Valencia, Spain (ECAI 2004)*. IOS Press, August 2004.

[5] K. Stathis, A.C. Kakas, W. Lu, N. Demetriou, U. Endriss, and A. Bracciali. PROSOCS: A platform for programming software agents in computational logic. *Applied Artificial Intelligence*, 20(4-5), 2006.

[6] F. Toni. Argumentative KGP agents for service composition. Proc. AITA08, Architectures for Intelligent Theory-Based Agents, AAAI Spring Symposium, March 2008, Stanford University, CA, USA, 2008.

[7] M. Morge, P. Mancarella, F. Toni, J. McGinnis, S. Bromuri, and K. Stathis Toward a modular architecture of argumentative agents to compose services. In Proceedings JFSMA 2007, 2007.

5 Contacts

Paolo Mancarella (Corresponding Author)
Dipartimento di Informatica, Università di Pisa
Largo B. Pontecorvo, 3
56127 Pisa, Italy
Tel: +39 050 2212 710
paolo.mancarella@unipi.it

Francesca Toni
Department of Computing, Imperial College London
South Kensington Campus, Huxley Building
London SW7 2AZ, UK
Tel: +44 (0)20 7594 8228
ft@doc.ic.ac.uk

LOGICHE NON-CLASSICHE PER LA RAPPRESENTAZIONE DELLA CONOSCENZA E IL RAGIONAMENTO
NON-CLASSICAL LOGICS FOR KNOWLEDGE REPRESENTATION AND REASONING

Laura Giordano, Valentina Gliozzi, Nicola Olivetti, Gian Luca Pozzato, and Camilla B. Schwind

SOMMARIO/*ABSTRACT*

Riassumiamo brevemente la nostra attività di ricerca nel campo delle logiche non-classiche iniziata negli anni '90. In particolare, descriviamo la nostra ricerca riguardante l'applicazione delle logiche non-classiche alla rappresentazione della conoscenza e lo sviluppo di metodi di prova per logiche non-monotone e condizionali.

We briefly outline our research activity in the field of non-classical logics started in the 90s. In particular, we describe our research in the application of non-classical logics to knowledge representation and in the development of proof methods for non-monotonic and conditional logics.

Keywords: Non-classical logics, knowledge representation, proof methods

1 Introduction

Our interest in the field of non-classical logic started with our work in Logic Programming at the beginning of the 90s. At that time we were working with Alberto on extensions of LP for dealing with hypothetical, conditional, defeasible and abductive reasoning. Those activities include the development of goal directed proof methods for Horn like fragments of modal logics K, S4, S5 and their use in the definition of structuring constructs for logic programs; the study of negation as failure in a hypothetical logic programming (NProlog); the semantic characterization of truth maintenance systems (TMS), and its relation with stable model semantics; proof procedures for abductive logic programming; and the definition of a conditional logic programming language (CondLP). Since that time, we have started working on non-classical logics both focusing on the use of such logics in knowledge representation and on developing proof methods for the automatization of conditional and non-monotonic logics.

Non-classical logics are widely used within the AI community, in the context of knowledge representation. In the following section, we describe the activity of our group in this area, concerning the use of modal, temporal, conditional and non-monotonic logics for Reasoning about Actions and Change and for Belief Revision as well as in the specification and verification of multi-agent systems.

In section 3 we describe our activity regarding proof methods for non-classical logics and, in particular, for KLM non-monotonic logics and for Conditional Logics.

2 Knowledge Representation

As mentioned above, our activity in Knowledge Representation has been mainly concerned with the formalization of *change*, which is crucial both in the context of Reasoning about Actions as well as in the context of Belief Revision. Concerning Reasoning about Actions, we have proposed a few modal and temporal formalisms for modelling actions execution. In modal and temporal action theories, action execution is modelled by introducing action modalities, and the Ramification problem is addressed by making use of modal or temporal operators (see section 2.1). Such action theories have been used in the specification and verification of agent interaction protocols as well as in the specification, verification and composition of web services (section 2.2). Concerning Belief Revision, our research has mainly focused on the relationships between Belief Revision and Conditional Logics (section 2.3). In the following we describe the above activities, as well as our recent activity concerning reasoning about typicality and inheritance with exceptions in Description Logics (section 2.4).

2.1 Reasoning About Actions

The idea of representing actions as modalities comes from Dynamic Logics [15]. As observed in [17], classical dynamic logic adopts essentially the same ontology as McCarthy's situation calculus, by taking "the state of the world as primary, and encoding actions as transformations on states". Indeed, actions can be represented in a natural way by modalities, and states as sequences of modalities. In this setting, the action law, saying that action a has effect f when executed in a state in which P holds, can be expressed by the formula: $P \rightarrow [a]f$. Moreover, the precondition law, saying that action a is executable in a state in which condition C holds, can be expressed by the formula: $C \rightarrow < a > f$. Based on this idea, in [10] we have defined a modal action theory in which the frame problem is tackled by using a non-monotonic formalism which maximizes persistency assumptions and the ramification problem is tackled by introducing a modal causality operator which is used to represent causal dependencies among fluents. This action theory can also deal with incomplete initial state and with nondeterministic actions.

In [10], we have developed a temporal action theory based on a dynamic extension of Linear Temporal Logic (LTL). This logic, called DLTL (Dynamic Linear Time Temporal Logic) [16], extends LTL by strengthening the "until" operator by indexing it with regular programs. The advantage of using a linear time temporal logic is that it is a well established formalism for specifying the behavior of distributed systems, for which a rich theory has been developed and the verification task can be automated by making use of automata based techniques. In particular, for DLTL, in [11] a tableau-based algorithm for obtaining a Büchi automaton from a formula in DLTL has been presented, whose construction can be done on-the-fly, while checking for the emptiness of the automaton.

An alternative approach to reasoning about actions, based on Conditional Logics, has been proposed in [14].

2.2 Specification and Verification of Agent Interaction Protocols

The temporal action theory described above has been used in the specification and verification of communication protocols [12]. We have followed a social approach [22] to agent communication, where communication is described in terms of changes to the social relations between participants, and protocols in terms of creation, manipulation and satisfaction of commitments among agents. The description of the interaction protocol and of communicative actions is given in a temporal action theory, and agent programs, when known, can be specified as complex actions (regular programs in DLTL).

We have addresses several kinds of verification problems, including run-time verification of protocols as well as static verification of agent compliance with the protocols. Some of these problems can be formalized either as validity or as satisfiability problems in the temporal logic and can be solved by model checking techniques. Other problems, as compliance, are more challenging and require a special treatment [13]. The proposed approach has also been used in the specification of Web Services and, in particular, for reasoning about service composition.

2.3 Belief Revision

A lot of work has been devoted to the problem of finding a formal relation between Conditional Logics and Belief Revision [4, 18]. Conditional Logics provide a semantics to conditional sentences of the form "if A, then B", denoted by $A \Rightarrow B$. Belief Revision is the area of Knowledge Representation that deals with the problem of how to integrate a new information in a given belief set. The most known theory of Belief Revision is the so-called AGM theory (from Alchourrón, Gardenfors, and Makinson who first proposed it) that specifies a set of rationality postulates for integrating a new information about a static domain into a belief set of the same domain.

The idea that there might be a relation between evaluation of conditional sentences and Belief Revision dates back to Ramsey, who proposed an acceptability criterion for conditionals in terms of belief change. According to this criterion, in order to decide whether to accept a conditional $A \Rightarrow B$ in a belief set K, one should add A to K by changing it as little as possible, and see if B follows. If it does, one should accept the conditional, otherwise one should reject it. In spite of the intuitiveness of Ramsey's criterion, its formalisation in the framework of Belief Revision is not straightforward. Many proposals, such as [4] run into the well-known Triviality Result, according to which there is no interesting Belief Revision system compatible with the proposed formalization. In [7, 8] we have proposed a Conditional Logic that corresponds to Belief Revision, thus establishing a relation between the two domains, without running into the Triviality Result.

2.4 Reasoning About Typicality in Description Logics

The family of description logics (DLs) is one of the most important formalisms of knowledge representation. DLs correspond to tractable fragments of first order logic, and are reminiscent of the early semantic networks and of frame-based systems. They offer two key advantages: a well-defined semantics based on first-order logic and a good trade-off between expressivity and complexity. DLs have been successfully implemented by a range of systems and they are at the base of languages for the semantic web such as OWL.

A DL knowledge base comprises two components: (i) the TBox, containing the definition of concepts (and possibly roles), and a specification of inclusions relations among them, and (ii) the ABox containing instances of concepts and roles, in other words, properties and relations of individuals. Since the very objective of the TBox is to build a taxonomy of concepts, the need of representing prototypical properties and of reasoning about defeasible inheritance of such properties naturally arises. The traditional approach is to handle defeasible inheritance by integrating some kind of non-monotonic reasoning mechanism. This has led to study non-monotonic extensions of

DLs. However, finding a suitable non-monotonic extension for inheritance reasoning with exceptions is far from obvious.

In [5], we have considered a novel approach to defeasible reasoning based on the use of a typicality operator $\mathbf{T}$. The intended meaning is that, for any concept C, $\mathbf{T}(C)$ singles out the instances of C that are considered as "typical" or "normal". Thus, an assertion as "normally students do not pay taxes" is represented by $\mathbf{T}(Student) \sqsubseteq \neg TaxPayer$. The DL obtained is called $\mathcal{ALC} + \mathbf{T}$.

In the logic $\mathcal{ALC} + \mathbf{T}$, one can have consistent knowledge bases containing the inclusions $\mathbf{T}(Student) \sqsubseteq \neg TaxPayer$; $\mathbf{T}(Student \sqcap Worker) \sqsubseteq TaxPayer$; $\mathbf{T}(Student \sqcap Worker \sqcap \exists HasChild.\top) \sqsubseteq \neg TaxPayer$, corresponding to the assertions: normally a student does not pay taxes, normally a working student pays taxes, but normally a working student having children does not pay taxes (because he is discharged by the government), etc.. Furthermore, if the ABox contains the information that for instance $\mathbf{T}(Student \sqcap Worker)(john)$, one can infer that $TaxPayer(john)$.

3 Proof Methods for Non-classical Logics

Our interest in the area of proof methods started with our work in Logic Programming

At the beginning of the Nineties, our interest for proof methods for non-classical logics were mainly devoted to extend goal directed proof methods to non-classical logics, and, in particular to modal logics. In the same period, Dale Miller [19] was putting the basis of intuitionistic logic programming, based on the idea of having uniform proofs. Our work in this field was mainly concerned with modal extensions of logic programmimg [1, 3] as well as with abductive, hypothetical and conditional extension of logic programming [2]. In the following, we describe our more recent activity concerning proof methods for non-monotonic and conditional logics.

3.1 Proof Methods for KLM Logics

In [9] we have introduced analytic tableau calculi for all non-monotonic logics introduced by Kraus, Lehmann, and Magidor (KLM). Such logics, namely **R**, **P**, **CL**, and **C**, have a preferential semantics in which a preference relation is defined among worlds or states. It has been observed that KLM logics correspond to the flat (i.e. unnested) fragment of well-known Conditional Logics.

Our tableau method provides a sort of run-time translation of **P** into modal logic G. The idea is simply to interpret the preference relation as an accessibility relation: a conditional $A \mid\!\sim B$ holds in a model if B is true in all minimal A-worlds, where a world w is an A-world if it satisfies A, and it is a minimal A-world if there is no A-world w' preferred to w. The relation with modal logic G is motivated by the fact that we assume, following KLM, the so-called *smoothness condition*, which ensures that minimal A-worlds exist whenever there are A-worlds, by preventing infinitely descending chains of worlds. This condition therefore corresponds to the finite-chain condition on the accessibility relation (as in modal logic G).

We have extended our approach to the cases of **CL** and **C** by using a second modality which takes care of states (intuitively, sets of worlds). Regarding **CL**, we have shown that we can map **CL**-models into **P**-models with an additional modality. In both cases, we can define a decision procedure to solve the validity problem in CoNP. Also, we have given a labelled calculus for the strongest logic **R**, where the preference relation is assumed to be modular. The calculus defines a systematic procedure which allows the satisfiability problem for **R** to be decided in nondeterministic polynomial time.

From the completeness of our calculi we get for free the finite model property for all the logics considered. With the exception of the calculus for **C**, in order to ensure termination, our tableau procedures for KLM logics do not need any loop-checking, nor blocking, nor caching machinery. Termination is ensured only by adopting a restriction on the order of application of the rules.

3.2 Proof Methods for Conditional Logics

In [20] we have introduced proof methods for some standard Conditional Logics. We have considered the *selection function* semantics. Intuitively, the selection function f selects, for a world w and a formula A, the set of worlds $f(w, A)$ which are "most similar to w" given the information A. In this respect, the selection function can be seen as a sort of modality indexed by formulas of the language. A conditional formula $A \Rightarrow B$ holds in a world w if B holds in all the worlds selected by f for w and A.

We have introduced cut-free sequent calculi for the basic Conditional Logic CK and for some of its extensions, namely CK+{ID, MP, CS, CEM} including all the combinations of these extensions except those including *both* CEM and MP. Our calculi make use of labels representing possible worlds. Two types of formulas are involved in the rules of the calculi: world formulas of the form $x : A$, representing that A holds at world x, and transition formulas of the form $x \xrightarrow{A} y$, representing that $y \in f(x, A)$. The completeness of the calculi is an immediate consequence of the admissibility of cut.

We have also shown that one can derive a decision procedure from the cut-free calculi. Whereas the decidability of these systems was already proved by Nute (by a finite-model property argument), our calculi give the first *constructive* proof of decidability. As usual, the terminating proof search mechanism is obtained by controlling the backward application of some critical rules. By estimating the size of the finite derivations of a given sequent, we have also obtained a polynomial space complexity bound for the logics considered.

Our calculi can be the starting point to define goaloriented proof procedures, according to the paradigm of Miller's Uniform Proofs recalled above. As a preliminary result, in [21] we have presented a goal-directed calculus for a fragment of CK and its extensions with MP and ID.

Proof methods for other Conditional Logics have been introduced in [6]. In detail, some labelled tableaux calculi have been defined for the Conditional Logic **CE** and its main extensions, including **CV**, whose flat fragment correspond, respectively, to KLM systems **P** and **R**.

4 Conclusions and Future Works

We believe that the temporal action theory we have developed for the specification and verification of agent interaction protocols can be profitably used in the specification and verification of web services. In this context, new issues arise, as for instance the problem of modelling service composition and that of service compliance (which still requires a general solution).

Concerning reasoning about typicality in description logics, we are currently studying a minimal model semantics for $\mathcal{ALC} + \mathbf{T}$ to maximize typical instances of a concept. By means of this semantics we are able to infer defeasible properties of (explicit or implicit) individuals.

REFERENCES

[1] M. Baldoni, L. Giordano, and A. Martelli. A modal extension of logic programming: Modularity, beliefs and hypothetical reasoning. *Journal of Logic and Computation*, 8(5):597–635, 1998.

[2] D.M. Gabbay, L. Giordano, A. Martelli, N. Olivetti and M.L. Sapino. Conditional reasoning in Logic Programming. *J. of Logic Programming*, 44(1-3):37–74, 2000.

[3] D.M. Gabbay and N. Olivetti. *Goal-Directed Proof Theory (Applied Logic Series V. 21)* , Springer, 2000.

[4] P. Gärdenfors. Belief Revisions and the Ramsey Test for Conditionals. *Philosoph. Review*, 95:81–93, 1986.

[5] L. Giordano, V. Gliozzi, N. Olivetti, and G.L. Pozzato. Preferential Descritpion Logics. In *LPAR 2007*. LNAI, 4790, pp. 257-272, Springer, 2007.

[6] L. Giordano, V. Gliozzi, N. Olivetti, and C.B. Schwind. Tableau Calculi for Preference-Based Conditional Logics. In *TABLEAUX 2003*, LNAI, 2796, pp. 81-101, Springer, 2003.

[7] L. Giordano, V. Gliozzi, and N. Olivetti. Iterated Belief Revision and Conditional Logic. *Studia Logica*, 70(1):23–47, 2002.

[8] L. Giordano, V. Gliozzi, and N. Olivetti. Weak AGM postulates and strong Ramsey Test: A logical formalization. *Artificial Intelligence*, 168(1-2):1–37, 2005.

[9] L. Giordano, V. Gliozzi, N. Olivetti, and G.L. Pozzato. Analytic Tableaux Calculi for KLM Logics of Nonmonotonic Reasoning. *ACM Transactions on Computational Logic (ToCL)*, to appear.

[10] L. Giordano, A. Martelli and C. Schwind. Ramification and causality in a modal action logic. *Journal of Logic and Computation*, 10(5):625–662, 2000.

[11] L. Giordano, A. Martelli. Tableau-Based Automata Construction for Dynamic Linear Time Temporal Logic. *Annals of Mathematics and Artificial Intelligence*, 46(3): 289-315 (2006).

[12] L. Giordano, A. Martelli, C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic, *J. of Applied Logic*, 5(2): 214-234 (2007).

[13] L. Giordano and A. Martelli. Verifying Agent Conformance with Protocols specified in a Temporal Action Logic. In *AI*IA 2007. Proceedings.*, LNAI, 4733, pp. 145-156, Springer, 2007.

[14] L. Giordano and C. Schwind. Conditional logic of actions and causation. *Artificial Intelligence*, 157(1-2):239–279, 2004.

[15] D. Harel. First order dynamic logic. In *Extensions of Classical Logic*, Handbook of Philosophical Logic II, pp. 497–604, 1984.

[16] J.G. Henriksen and P.S. Thiagarajan, Dynamic Linear Time Temporal Logic. *Annals of Pure and Applied logic*, 96(1-3):187–207, 1999.

[17] L.T. McCarty. Modalities over actions, I. model theory. *KR '94, Proceedings*, pp. 437–448, 1994.

[18] D. Makinson. The Gärdenfors impossibility theorem in non-monotonic contexts. *Studia Logica*, 49(1):1–6, 1990.

[19] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform Proofs as a Foundation for Logic Programming. *Annals of Pure and Applied Logic*, 51(1-2):125–157, 1991.

[20] N. Olivetti, G.L. Pozzato, and C.B. Schwind. A Sequent Calculus and a Theorem Prover for Standard Conditional Logics. *ACM Transactions on Computational Logic (ToCL)*, 8(4):22/1–22/51, 2007.

[21] N. Olivetti and G.L. Pozzato. Theorem Proving for Conditional Logics: CondLean and GOALD$\mathcal{U}$CK. *J. of Applied Non-Classical Logics (JANCL)*, to appear.

[22] M. P. Singh. A Social Semantics for Agent Communication Languages. *Issues in Agent Communication 2000*, pp. 31-45.

5 Contacts

Laura Giordano
Dipartimento di Informatica - Università del Piemonte Orientale “A. Avogadro”
via Bellini 25/G - 15100 Alessandria - Italy
Email: laura@mfn.unipmn.it

Valentina Gliozzi
Dipartimento di Informatica - Università degli Studi di Torino - Italy
corso Svizzera 185 - 10149 Turin
Email: gliozzi@di.unito.it

Nicola Olivetti
LSIS - UMR CNRS 6168 Université Paul Cézanne (Aix-Marseille 3) - France
Avenue Escadrille Normandie-Niemen 13397 Marseille Cedex 20
Email: nicola.olivetti@univ.u-3mrs.fr - *nicola.olivetti@lsis.org*

Gian Luca Pozzato
Dipartimento di Informatica - Università degli Studi di Torino - Italy
corso Svizzera 185 - 10149 Turin
Email: pozzato@di.unito.it

Camilla B. Schwind
École d’Architecture de Marseille - Luminy - France
184 avenue de Luminy - 13288 Marseille Cedex 9
Email: Camilla.Schwind@map.archi.fr

L. Giordano got the Ph.D. in Computer Science from the Università degli Studi di Torino in 1993. Since 1998 she is Professore Associato at the Facoltà di Scienze Matematiche, Fisiche e Naturali, Università del Piemonte Orientale - Amedeo Avogadro. Her research interests include: Non-monotonic Reasoning, Belief Revision, Reasoning about Action and Change, Multiagent Systems, Proof Methods for non-classical logics.

V. Gliozzi graduated in Philosophy at the Università degli Studi di Torino in 1997, and she got the Ph.D. in Computer Science from the same university in 2002 (with a thesis on Belief Revision and Conditional Logics). Since 2005 she is a researcher at the Department of Computer Science at Università di Torino. Her main research interests include logic, knowledge representation, non-classical logics.

N. Olivetti got the Ph.D. in Computer Science from the Università degli Studi di Torino in 1995. He is a Professor of Computer Science at the Paul Cézanne University (Aix-Marseille, France), and he is a member of the CNRS laboratory LSIS. His main research interests are automated deduction for non-classical logics (conditional, substructural, and many-valued logics), foundation and proof-theory of non-monotonic reasoning, extensions of logic programming, and Belief Revision.

G.L. Pozzato was born in Moncalieri (Turin) in 1978. He took his “Laurea” degree “summa cum laude” in Computer Science in 2003, and his Ph.D. in Computer Science in 2007, both at the Università degli Studi di Torino. Since 2007 he is a researcher at the Department of Computer Science of the Università degli Studi di Torino. His research interests include non-monotonic reasoning, non-classical logics, proof-theory, and Description Logics.

C. B. Schwind is a researcher of Computer Science at the CNRS laboratory LIF (Marseille, France). Her main research interests are: Conditional Logic, Multi-agent systems, and analytic tableaux for non-monotonic and conditional logics. She has also been actively involved in basic research in the following topics: Natural Language Understanding, Temporal and Modal Logics, Deductive Data Bases, Computer Assisted Language Learning, Action Logics and the Frame Problem, Modal Non Monotonic Logics.

Logiche multimodali per ragionare sull'interazione
Multimodal Logics for Reasoning about Interaction

Matteo Baldoni, Cristina Baroglio and Viviana Patti

SOMMARIO/*ABSTRACT*

Questo articolo presenta alcune delle attività condotte nel corso degli ultimi anni dal gruppo di ricerca guidato da Alberto Martelli. In particolare verrà presentato un percorso che comprende la specifica, lo sviluppo e la verifica di protocolli di interazione. Il filo conduttore è costituito dall'uso di logiche multimodali e di formalismi dichiarativi e tecniche di ragionamento basati sulla logica computazionale.

In this paper, we report some of the activities carried on in the last years by the research group leaded by Alberto Martelli. In particular, it presents a research line that encompasses the specification, the development and the verification of interaction protocols. The leading thread is given by the use of multimodal logics and of declarative formalisms and reasoning techniques, based on computational logic.

Keywords: Interaction protocols, multimodal logics, web services, semantic web.

1 Introduction

Modal logics are widely used in Artificial Intelligence for representing *knowledge* and *beliefs* together with other attitudes like, for instance, *goals*, *intentions* and *obligations*. Moreover, modal logics are well suited for representing *dynamic* aspects in *agent systems* and, in particular, to formalize reasoning about *actions* and *time* [16, 22]. In this context, one of the main research lines of the last years concerns the specification of interaction and the forms of reasoning that can be applied to it, and gives particular attention to the verification of properties of the interaction and of the interacting agents [22]. The work that we summarize in these pages begins with the construction of a logical framework, based on a class of normal multimodal logic (called *grammar logics*). This framework has a computational counterpart, which is particularly suitable for representing the behavior of interacting and communicative agents and which lead to the implementation of the declarative programming language Dynamics in LOGic. The framework has been successfully applied to as diverse application domains as web-based adaptive tutoring, web service selection and composition, reasoning about choreographies, semantic web. In particular, it has been adopted in various national and international research projects, e.g. MASSiVE, SVP, and REWERSE.

Future directions

Declarative languages are becoming very important in the context of Semantic Web, especially in the most recent years, when the focus moved from the ontology layer to the logic layer and the need of expressing rules and apply various forms of reasoning have emerged. This interest is witnessed also by the creation of a W3C working group to define a Rule Interchange Format. The effort done for representing and reasoning about interactions, in the framework presented in this paper, finds a natural grounding in the development of negotiation or personalization policies, expressed by rules. Challenging applications can be identified also in the context of web services. Here, we are interested in applications aimed at fostering the re-use of software, task that requires abilities, e.g. flexibility, which are supplied by declarative languages and by the reasoning techniques that they allow to apply. In particular, a very promising direction of research is the study of methods and approaches to verify the interoperability of services and the conformance of a service to a choreography role.

2 The origins: A class of Normal Multimodal Logics

In [12, 3] a class of *normal multimodal logics*, called *grammar logics*, is studied. The class is characterized by a set of logical axioms of the form:

$$[p_1] \dots [p_n]\varphi \supset [s_1] \dots [s_m]\varphi \quad (n > 0 \ m \geq 0) \qquad (1)$$

called *inclusion axioms*, where the p_i's and s_j's are modalities. This class includes some well-known modal systems such as K, $K4$, $S4$ and their multimodal versions. Differently from other logics, such as those in [16], these systems can be *non-homogeneous* (i.e., every modal operator is not restricted to belong to the same system) and can contain some *interaction axioms* (i.e., modal operators are not restricted to be mutually independent).

This class of logics has been introduced by Fariñas del Cerro and Penttonen in [14] to simulate the behavior of *formal grammars*. Given a formal grammar, a modality is associated to each terminal and nonterminal symbol, while, for each production rule of the form $p_1 \cdots p_n \rightarrow s_1 \cdots s_m$, an inclusion axiom $[p_1] \dots [p_n]\varphi \supset [s_1] \dots [s_m]\varphi$ is defined. In [14] it is shown that testing whether a word is generated by the formal grammar is equivalent to proving a theorem in the logic, thus showing the *undecidability* of the whole class of logics.

In [12, 3] an *analytic tableau calculus* for the class of *grammar logics* is presented. The calculus is parametric w.r.t. the logics of this class. In particular, they deal with *non-homogeneous* multimodal systems with arbitrary *interaction axioms* of form (1). The calculus is a prefixed tableaux extension of those in [18, 15]. Prefixes are given an atomic name and the accessibility relationships among them are explicitly represented in a graph. The key idea is using the characterizing axioms of the logic as "*rewrite rules*" which create new paths among worlds in the counter-model construction. The works prove the *undecidability* of modal systems based on *context-sensitive* and *context-free* grammars, while *right regular* grammars are *decidable* (by using an extension of the *filtration methods* by the Fischer-Ladner closure for modal logics). In the particular case when m is 1, the axiom schema reduces to

$$\langle s_0 \rangle\varphi \subset \langle t_1 \rangle\langle t_2 \rangle \dots \langle t_n \rangle\varphi \qquad (2)$$

and the rewriting rules for describing accessibility relations become similar to a Prolog goal-directed proof procedure. This observation has allowed the definition of the language Dynamics in LOGic. This class of logics has also been used in the study of description logics [20, 17] and extended to more general forms of interaction in [4].

3 Dynamics in LOGic: An agent Programming Language

Dynamics in LOGic [13] has been developed as a language for programming agents and is based on a logical theory for reasoning about actions and change in a modal logic programming setting. An agent's behavior is described in a non-deterministic way by giving the set of actions that it can perform. Specifically, it can be specified by a *domain description*, which includes: a) *action and precondition laws*, describing the atomic world actions the agent may perform; b) a set of *sensing axioms* describing the agent's atomic sensing actions; c) a set of *procedure axioms* describing the agent's complex behavior. Each atomic action can have preconditions to its application (that decide if the action is executable) and effects due to its application. Moreover, effects can be subject to further conditions in order to become true. For instance, the executability precondition to the action "paying by credit card" is that I hold a valid credit card. A conditional effect of this action could be "to be notified by SMS about payments". This effect will become true only if I subscribed the service (precondition to the effect).

Given this view of actions, we can think to the problem of reasoning as the act of building or of traversing a sequence of transitions between *states*. Technically speaking, a state is a set of *fluents*, i.e., properties whose truth value can change over time. Such properties encode the information that flows during the execution of a policy: for instance, if a requester communicates to pay by miles, this information will be included in the state of the provider as a fluent. In general, we cannot assume that the value of each fluent in a state is known, so we want to have both the possibility of representing that some fluents are unknown and the ability of reasoning about the execution of actions on incomplete states. To explicitly represent the unknown value of some fluents, we introduced an epistemic operator $\mathcal{B}$, to represent the beliefs an entity has about the world: $\mathcal{B}f$ means that the fluent f is known to be true, $\mathcal{B}\neg f$ means that the fluent f is known to be false. A fluent f is undefined when both $\neg\mathcal{B}f$ and $\neg\mathcal{B}\neg f$ hold. Thus each fluent in a state can have one of the three values: *true*, *false* or *unknown*.

Complex behaviors can be specified by means of *procedures*, Prolog-like clauses built upon other actions. Formally, a complex action is a collection of *inclusion axiom schemas* of the modal logic, of form (2). s_0 is a *procedure name* and the p_i's are *procedure names*, *atomic actions*, or *test actions* (f). Procedure definitions may be *recursive* and procedure clauses can be executed in a *goal-directed* way, similarly to standard logic programs.

The language allows the specification of communicative behaviors [10]. Indeed, we define the *communication kit* [19] for an agent as consisting of a predefined set of primitive speech acts the agent can perform/recognize, modeled in terms of action and preconditions laws, a set of special sensing actions for getting new information by external communications, defined by sensing axioms, and a set of interaction protocols specified by procedure axioms. Usually a communicative action modifies not only the beliefs of the executor about the world but also its beliefs about the interlocutor's mental state.

Given a domain description, we can reason about it and formalize the *temporal projection* and the *planning* problems by means of existential queries of form:

$$\langle p_1\rangle\langle p_2\rangle \ldots \langle p_m\rangle Fs \qquad (3)$$

where each p_k, $k = 1, \ldots, m$ may be an (atomic or complex) action executed by the agent. Checking if a query of form (3) succeeds corresponds to answering the question "Is there an execution trace of the sequence p_1, ..., p_m that leads to a state where the conjunction of belief fluents *Fs* holds for our agent?". In case all the p_k's are atomic actions, it amounts to predict if the condition of interest will be true after their execution. In case complex actions are involved, the execution trace that is returned in the end is a *plan* to bring about *Fs*. The procedure definition constrains the search space. The plan can be conditional because whenever a sensing action is involved, none of the possible answers of the interlocutor can be excluded.

A goal-directed proof procedure, based on negation as failure, allows to (dis)prove queries of form (3). An interpreter for the language has been implemented in Sicstus Prolog [8]. This implementation allows the language to be used as an ordinary programming language for *executing* procedures which specify the behavior of an agent, but also for *reasoning* about them, by extracting linear or conditional plans. The plan extraction process of the interpreter is a straightforward implementation of the proof procedure contained in the theoretical specification of the language.

3.1 Web-based Adaptive Tutoring

Dynamics in LOGic has been used to implement an adaptive tutoring system [11] with a multi-agent architecture, that can produce *personalized study* plans and that can validate study plans built by a user. A key feature that allows the tutoring system agents to *adapt to users* is their ability to tackle mental attitudes, such as beliefs and intentions. The agent can adopt the user's learning goal and find a way for achieving it, which fits the specific student's interests and takes into account his/her current knowledge. A natural evolution of this work opened the way to the activity carried on in the REWERSE network of excellence [2, 9, 7].

4 Reasoning about WS Composition and Choreographies

In the last years distributed applications over the World-Wide Web have obtained wide popularity and uniform mechanisms have been developed for handling computing problems which involve a large number of heterogeneous components, that are physically distributed and that interoperate. These developments have begun to coalesce around the web service paradigm, where a service can be seen as a component available over the web [1]. Each service has an interface that is accessible through standard protocols and that describes the interaction capabilities of the service.

The work presented in [10] faces the problem of automatic selection and composition of web services, discussing the advantages that derive from the inclusion, in a web service declarative description, of the high-level interaction protocol, that is used by the service for interacting with its partners, allowing a rational inspection of it. Communication can, in fact, be considered as the behavior resulting from the application of a special kind of actions: communication actions. The reasoning problem that this proposal faces can intuitively be described as looking for an answer to the question "Is it possible to make a deal with this service respecting the user's goals?". Given a logic-based representation of the service policies and a representation of the customer's needs as abstract goals, expressed by a logic formula, logic programming reasoning techniques are used for understanding if the constraints of the customer fit in with the policy of the service.

In this issue it is possible to distinguish three necessary components: (*i*) web services capabilities must be represented according to some declarative formalism with a well-defined semantics, as also observed by van der Aalst et al. [21]; (*ii*) automated tools for reasoning about such a description and performing tasks of interest must be developed; (*iii*) in order to gain flexibility in fulfilling the user's request, reasoning tools should represent such requests as *abstract goals*.

Our proposal is set in the Semantic Web field of research and inherits from research in the field of multi-agent systems. In particular, the declarative descriptions of services are based on the modal logic programming framework introduced in Section 3. Web services are viewed as software agents, communicating by predefined sharable interaction protocols, where the protocol-based interactions are formalized as Dynamics in LOGic procedures; reasoning about actions and change techniques (planning) are used for performing the selection and composition of web services in a way that is personalized w.r.t. the user's request. Applying reasoning techniques on a declarative specification of the service interactions allows to gain flexibility in *fulfilling the user preference* in the context of a web service matchmaking process. As a quick example, consider a web service that allows buying products, alternatively paying cash or by credit card: a user might have preferences on the form of payment to enact. In order to decide whether or not buying at this shop, it is necessary to single out the specific course of interaction that allows buying cash. This form of personalization requires to reason about a description of the service interaction policy.

A declarative specification of the interaction is useful also in the process of selecting the services which will play the various roles of the given choreography, in the particular case in which a condition of interest is to be preserved (the goal for which the service is sought). In [6, 5] we show that current semantic matchmaking techniques do not guarantee goal preservation. The approach is based on an action-based representation of the operations of a service: each operation is described in terms of its preconditions and effects. Also in this work, the Dynamics in LOGic framework was used to represent service interaction policies as well as roles. This representation allow to reason for checking if it is possible to reach a goal by adopting a certain role, and if the goal is preserved after the substitution of the service capabilities to the abstract requirements specified in the role. We show that, by exploiting reasoning mechanisms and the choreography definition, it is possible to overcome the limits of the current semantic matchmaking techniques and we have proposed a variant of the *plugin match* which guarantees goal preservation.

5 Acknowledgements

The authors would like to thank all people who contributed to this line of research over the last years, in particular Alberto Martelli, Alessandro Chiarotto, Laura Giordano, Claudio Schifanella and Laura Torasso, co-authoring most of the papers.

REFERENCES

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer, 2004.

[2] G. Antoniou, M. Baldoni, C. Baroglio, R. Baungartner, F. Bry, T. Eiter, N. Henze, M. Herzog, W. May, V. Patti, S. Schaffert, R. Schidlauer, and H. Tompits. Reasoning Methods for Personalization on the Semantic Web. *Ann. of Math., Comp. & Teleinf. (AMCT)*, 2(1):1–24, 2004.

[3] M. Baldoni. *Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension*. PhD thesis, Università degli Studi di Torino, Italy, 1998.

[4] M. Baldoni. Normal Multimodal Logics with Interaction Axioms. In *Labelled Deduction*, *Applied Logic Series* 17, pp. 33–53. Kluwer Ac. Publisher, 2000.

[5] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Reasoning on choreographies and capability requirements. *Int. J. of Business Process Integration and Management*, 2(4), 2007.

[6] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella. Service selection by choreography-driven matching. In *Proc. of WEWST'07*, vol. 313 of *CEUR, Workshop Proc.*, pp. 1–17, 2008.

[7] M. Baldoni, C. Baroglio, I. Brunkhorst, E .Marengo, and V. Patti. Reasoning-based curriculum sequencing and validation: Integration in a service-oriented architecture. In *EC-TEL*, *LNCS* 4753, pp. 426–431. Springer, 2007.

[8] M .Baldoni, C. Baroglio, A. Chiarotto, and V. Patti. Programming goal-driven web sites using an agent logic language. In *PADL*, *LNCS* 1990, pp. 60–75. Springer, 2001.

[9] M. Baldoni, C. Baroglio, and N. Henze. Personalization for the semantic web. In *Reasoning Web*, *LNCS* 3564, pp. 173–212. Springer, 2005.

[10] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti. Reasoning about interaction protocols for customizing web service selection and composition. *J. Log. Algebr. Program.*, 70(1):53–73, 2007.

[11] M. Baldoni, C. Baroglio, and V. Patti. Web-based adaptive tutoring: An approach based on logic agents and reasoning about actions. *Artif. Intell. Rev.*, 22(1):3–39, 2004.

[12] M. Baldoni, L. Giordano, and A. Martelli. A tableau for multimodal logics and some (un)decidability results. In *TABLEAUX*, *LNCS* 1397, pp. 44–59. Springer, 1998.

[13] M. Baldoni, A. Martelli, V. Patti, and L. Giordano. Programming rational agents in a modal action logic. *Ann. Math. Artif. Intell.*, 41(2-4):207–257, 2004.

[14] L. Fariñas del Cerro and M. Penttonen. Grammar Logics. *Logique et Analyse*, 121-122:123–134, 1988.

[15] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics*, volume 169 of *Synthese library*. D. Reidel, Dordrecht, Holland, 1983.

[16] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artif. Intell.*, 54(2):319–379, 1992.

[17] Ian Horrocks and Ulrike Sattler. Decidability of shiq with complex role inclusion axioms. *Artif. Intell.*, 160(1-2):79–104, 2004.

[18] A. Nerode. Some Lectures on Modal Logic. In F. L. Bauer, editor, *Logic, Algebra, and Computation*, volume 79 of *NATO ASI Series*. Springer-Verlag, 1989.

[19] V. Patti. *Programming Rational Agents: a Modal Approach in a Logic Programming Setting*. PhD thesis, Università degli Studi di Torino, Italy, 2002.

[20] Ulrike Sattler. Description Logics for Ontologies. In *Proc. of ICCS 2003*, *LNAI* 2746, pap. 96–116, 2003.

[21] Wil M. P. van der Aalst, Marlon Dumas, Arthur H. M. ter Hofstede, Nick Russell, H. M. W. (Eric) Verbeek, and Petia Wohed. Life after bpel? In *EPEW/WS-FM*, *LNCS* 3670, pp. 35–50. Springer, 2005.

[22] Michael Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.

6 Contacts

Matteo Baldoni (Corresponding Author)
Dipartimento di Informatica — Università degli Studi di Torino
c.so Svizzera, 185
I-10149 Torino (Italy)
Tel. +39 011 6706756
baldoni@di.unito.it

Cristina Baroglio
Dipartimento di Informatica — Università degli Studi di Torino
c.so Svizzera, 185
I-10149 Torino (Italy)
Tel. +39 011 6706703
baroglio@di.unito.it

Viviana Patti
Dipartimento di Informatica — Università degli Studi di Torino
c.so Svizzera, 185
I-10149 Torino (Italy)
Tel. +39 011 6706804
patti@di.unito.it

7 Biography

Matteo Baldoni (http://www.di.unito.it/ baldoni) is associate professor at the University of Torino since 2006. He received a Ph.D. in Computer Science from the same university. He has a background in computational logic, multi-modal and non-monotonic extensions of logic programming and reasoning about actions. Current research interests: communication protocol design and implementation, conformance and interoperability of services, personalization by reasoning in the semantic web. He is co-chair of the workshop AWESOME@MALLOW'007, and has been co-chair of the last four editions of the DALT@AAMAS international workshop. He chairs the working group "Sistemi ad Agenti e Multiagente" of the Italian Association for Artificial Intelligence.

Cristina Baroglio (http://www.di.unito.it/ baroglio) is associate professor of Computer Science at the University of Torino since 2005. She has a Ph.D. in Cognitive Sciences from the same university, and has a background in Machine Learning and Automated Reasoning. She is author of over 70 papers, co-chair of the workshop "Agents, Web Services, and Ontologies: Integrated Methodologies" (AWESOME@MALLOW'007), and member of the PC of the Reasoning web Summer School 2007 and 2008. Her research interests include: semantic web and semantic web services, adaptation based on reasoning, conformance and interoperability of services to choreographies/protocols, automatic teaching to artificial agents, formal approaches to e-learning.

Viviana Patti (http://www.di.unito.it/ patti) is a researcher associate in Computer Science at the University of Torino since 2005. She received her Ph.D. in Computer Science in 2002. She is author of more than 50 scientific papers. Her research interests include: computational logics for agent programming, reasoning enabling personalization in the semantic web, web service interoperability and conformance verification, web-based education courseware and curricula. She has been involved in the organization of several international events mainly in the fields "Personalization in the Semantic Web", "Web Information systems and Technologies" and "Agents". She has been member of the European Network of Excellence REWERSE.

Authors Index