Appendix A of the Introduction to the 31st International Conference on Logic Programming Special Issue: Abstracts of Technical Communications

submitted 27 April 2015; accepted 5 June 2015; revised 21 July 2015

Parallel Bottom-Up Evaluation of Logic Programs: $\mathcal{DeALS}$ on Shared-Memory Multicore Machines

M. Yang, A. Shkapsky and C. Zaniolo

University of California, Los Angeles, USA
(*e-mail:* {yang,shkapsky,zaniolo}@cs.ucla.edu)

Delivering superior expressive power over RDBMS, while maintaining competitive performance, has represented the main goal and technical challenge for deductive database research since its inception forty years ago. Significant progress toward this ambitious goal is being achieved by the $\mathcal{DeALS}$ system through the parallel bottom-up evaluation of logic programs, including recursive programs with monotonic aggregates, on a shared-memory multicore machine.

In $\mathcal{DeALS}$, a program is represented as an AND/OR tree, where the parallel evaluation instantiates multiple copies of the same AND/OR tree that access the tables in the database concurrently. Synchronization methods such as locks are used to ensure the correctness of the evaluation. We describe a technique which finds an efficient hash partitioning strategy of the tables that minimizes the use of locks during the evaluation. Experimental results demonstrate the effectiveness of the proposed technique — $\mathcal{DeALS}$ achieves competitive performance on non-recursive programs compared with commercial RDBMSs and superior performance on recursive programs compared with other existing systems.

KEYWORDS: Parallel, Bottom-up Evaluation, Datalog, Multicore, AND/OR Tree

Grid Mind: Prolog-Based Simulation Environment for Future Energy Grids

Jan Rosecky[1,2], *Filip Prochazka*[2] *and Barbora Buhnova*[1]

[1] *Faculty of Informatics, Masaryk University, Brno, Czech Republic*
(*e-mail:* {j.rosecky,buhnova}@mail.muni.cz)
[2] *Mycroft Mind, a. s., Brno, Czech Republic*
(*e-mail:* {jan.rosecky,filip.prochazka}@mycroftmind.com)

Fundamental changes in the current energy grids, towards the so called smart grids, initiated a range of projects involving extensive deployment of metering and control devices into the grid infrastructure. Since in many countries, the choice of supportive information and communication technologies (ICT) for the grid devices still remains an open question, benchmarking tools aimed at predicting their behavior in the deployed solution play an essential role in the decision-making process.

This paper presents a Prolog-based simulation environment, named Grid Mind, primarily intended for the very purpose. The tool was successfully used to generate simulation scenarios in several smart-grid related projects and became a self-standing simulation tool for the evaluation of information and communication technologies used to deliver low-voltage metering and monitoring data. The tool is continuously evolving, aimed to become an integral part of the future energy grid design in the Czech Republic and beyond.

KEYWORDS: Simulation Environment; Smart Grid; Communication and Networking; ICT; Prolog

4

Logic Programming and Bisimulation[1]

Agostino Dovier

University of Udine, DIMI, ITALY
(*e-mail:* agostino.dovier@uniud.it)

The logic programming encoding of the set-theoretic graph property known as *bisimulation* is analyzed. This notion is of central importance in non-well-founded set theory, semantics of concurrency, model checking, and coinductive reasoning. From a modeling point of view, it is particularly interesting since it allows two alternative high-level characterizations. We analyze the encoding style of these modelings in various dialects of Logic Programming. Moreover, the notion also admits a polynomial-time maximum fixpoint procedure that we implemented in Prolog. Similar graph problems which are instead NP hard or not yet perfectly classified (e.g., graph isomorphism) can inherit most from the declarative encodings presented.

KEYWORDS: Logic Programming Modeling, Bisimulation

[1] The work is partially supported by INdAM GNCS 2014 and 2015 projects.

Parallel Execution of the ASP Computation - an Investigation on GPUs[2]

Agostino Dovier[1], *Andrea Formisano*[2], *Enrico Pontelli*[3], *Flavio Vella*[4]

[1]*Dip. di Matematica e Informatica, Università di Udine*
(*e-mail:* agostino.dovier@uniud.it)
[2]*Dip. di Matematica e Informatica, Università di Perugia*
(*e-mail:* formis@dmi.unipg.it)
[3]*Dept. of Computer Science, New Mexico State University*
(*e-mail:* epontell@cs.nmsu.edu)
[4]*IAC-CNR and Dip. di Informatica, Sapienza Università di Roma*
(*e-mail:* vella@di.uniroma1.it)

This paper illustrates the design and implementation of a conflict-driven ASP solver that is capable of exploiting the *Single-Instruction Multiple-Thread* parallelism offered by *General Purpose Graphical Processing Units (GPUs).* Modern GPUs are multi-core platforms, providing access to large number of cores at a very low cost, but at the price of a complex architecture with non-trivial synchronization and communication costs. The search strategy of the ASP solver follows the notion of *ASP computation*, that avoids the generation of unfounded sets. Conflict analysis and learning are also implemented to help the search. The CPU is used only to pre-process the program and to output the results. All the solving components, i.e., nogoods management, search strategy, (non-chronological) backjumping, heuristics, conflict analysis and learning, and unit propagation, are performed on the GPU by exploiting SIMT parallelism. The preliminary experimental results confirm the feasibility and scalability of the approach, and the potential to enhance performance of ASP solvers.

KEYWORDS: ASP Solvers, ASP Computation, SIMT Parallelism, GPU Computing

[2] Research partially supported by INdAM GNCS-14, GNCS-15 projects and NSF grants DBI-1458595, HRD-1345232, and DGE-0947465. Hardware partially supported by NVIDIA. We thank Massimiliano Fatica for the access to the Titan cards and Alessandro Dal Palù for the useful discussions.

6

On Type-directed Generation of Lambda Terms

Paul Tarau

Department of Computer Science and Engineering, University of North Texas, USA
(*e-mail:* paul.tarau@unt.edu)

We describe a Prolog-based combined lambda term generator and type-inferrer for closed well-typed terms of a given size, in de Bruijn notation. By interleaving term generation and type inference, as shown in the code below, we significantly reduce the (super-exponential) effort needed to generate closed terms and then filter the simply-typed ones among them.

```
generateTypedTerm(Size,Term,Type) :- genTyped(Term,Type,[],Size,0).

genTyped(v(I),V,Vs) --> {nth0(I,Vs,V0), unify_with_occurs_check(V,V0)}.
genTyped(a(A,B),Y,Vs) --> down, genTyped(A,(X->Y),Vs), genTyped(B,X,Vs).
genTyped(l(A),(X->Y),Vs) --> down, genTyped(A,Y,[X|Vs]).

down(From,To):-From>0,To is From-1.
```

By taking advantage of Prolog's unique bidirectional execution model and sound unification algorithm, our generator can build "customized" closed terms of a given type. This relational view of terms and their types enables the discovery of interesting patterns about frequently used type expressions occurring in well-typed functional programs. At the same time, our study uncovers the most "popular" types that govern function applications among a about a million small-sized lambda terms and hints toward practical uses to combinatorial software testing.

The paper also shows the effectiveness of Prolog as a meta-language for modeling properties of lambda terms and their types. Together with (Tarau 2015b; Tarau 2015a) it provides a logic programming-based declarative playground for experimenting with lambda terms, combinators, as well as their type inference and evaluation mechanisms.

References

Tarau, P. 2015a. On a Uniform Representation of Combinators, Arithmetic, Lambda Terms and Types. In *PPDP'15: Proceedings of the 17th international ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, E. Albert, Ed. ACM, New York, NY, USA, 244–255.

Tarau, P. 2015b. On Logic Programming Representations of Lambda Terms: de Bruijn Indices, Compression, Type Inference, Combinatorial Generation, Normalization. In *Proceedings of the Seventeenth International Symposium on Practical Aspects of Declarative Languages PADL'15*, E. Pontelli and T. C. Son, Eds. Springer, LNCS 8131, Portland, Oregon, USA, 115–131.

KEYWORDS: Lambda Calculus, de Bruijn Notation, Type Inference, Generation of Closed Simply-typed Lambda Terms, Logic Programming as a Meta-language.

Answer Set Application Programming: a Case Study on Tetris[3]

Peter Schüller[1] *and Antonius Weinzierl*[2]

[1] *Department of Computer Engineering, Faculty of Engineering, Marmara University*
(*e-mail:* peter.schuller@marmara.edu.tr)
[2] *Institute of Information Systems, TU Wien*
(*e-mail:* weinzierl@kr.tuwien.ac.at)

Answer-Set Programming (ASP) is a successful branch of the logic programming paradigm with many applications in modelling and solving of NP-hard problems. Combinatorial problems are the main application domain of ASP and it seems unsuitable for serving as a programming language for interactive applications. However, we conjecture that there is no theoretical obstacle for using ASP to that end. As witnessed by functional programming, it can be useful to use a declarative paradigm for creating applications. In this work we explore possibilities, benefits, and drawbacks, of programming an interactive application in ASP. We find that this is hard mainly for the following reasons: managing change over time, interaction with the user, generating output that is ordered (i.e., not a set), handling persistence of certain data, and ensuring efficiency. ASP and related fields provide powerful techniques for representing actions and change, executing programs with respect to external environments, and processing external events. Even if the full power of these techniques is not required to build an interactive application, combining them is necessary, and putting together these concepts in a practical framework is challenging. We realize such an integration in a framework we call Answer Set Application Programming framework which is based on the HEX language and features syntactic shortcuts to make application programming more intuitive. We describe design decisions and discuss alternative possibilities. Our sample application is a playable version of Tetris which demonstrates that ASP can be used as a general-purpose programming-language.

KEYWORDS: Answer-Set Programming, Programming Techniques, Knowledge Representation, Software Engineering, Nonmonotonic Reasoning

[3] This work has been supported by the Austrian Science Fund (FWF) Project P27730 and the Scientific and Technological Research Council of Turkey (TUBITAK) Grant 114E430.

Structural Resolution for Logic Programming

Patricia Johann[1] *and Ekaterina Komendantskaya*[2] *and Vladimir Komendantskiy*[3]

[1] *Department of Computer Science, Appalachian State University, USA*
(*e-mail:* johannp@appstate.edu)
[2] *School of Computing, University of Dundee, UK*
(*e-mail:* katya@computing.dundee.ac.uk)
[3] *Moixa, UK*
(*e-mail:* vladimir@moixaenergy.com)

As ICLP is celebrating the 200^{th} anniversary of George Boole, we are reflecting on the fundamental "laws" underlying derivations in logic programming (LP), and making an attempt to formulate some fundamental principles for first-order proof search, analogous in generality to Boole's "laws of thought" for propositional logic.

Any such principles must be able to reflect two important features of first-order proof search in LP: its recursive and non-deterministic nature. For this they must satisfy two criteria: to be able to (a) model infinite structures and (b) reflect the non-determinism of proof search, relating "laws of infinity" with "laws of non-determinism" in LP. To be implementable in practice, such principles also need to (c) model infinite derivations in some constructive, observational way, thus defining "laws of observability" for LP proof search.

We draw our inspiration from the classical approach to semantics of first-order logic and recursive schemes. Best summarised in *"Fundamental Properties of Infinite Trees"* (Courcelle, 1983) this approach models first-order terms as trees, defining a term tree as a map from a (finite or infinite) tree language to first-order signature. The definition is subject to laws relating the structure of the tree language to the structure of the first-order signature.

We extend this elegant theory of infinite trees to give an operational semantics of LP that satisfies criteria (a), (b), and (c) above. We introduce a *Three Tier Tree Calculus* (T^3C) that defines in a systematic way the three tiers of tree structures underlying proof search in LP: term trees as described above, rewriting trees whose codomains are given by term trees, and derivation trees whose codomains are given by rewriting trees.

In all three cases, we identify structural laws that hold in every tier irrespective of the size of the tree language constituting the domain. We thus formulate structural (constructive) laws for finite and infinite tree structures arising in LP proof search that satisfy criteria (a) and (c). We further show how non-deterministic nature of LP proof search can be modeled by the new formalism of T^3C, so that it satisfies criterion (b) as well. Overall this shows that T^3C defines a new — *structural* — version of resolution for LP.

KEYWORDS: Structural Resolution, Term Trees, Rewriting Trees, Derivation Trees

Debugging ASP using ILP

Tingting Li[1], *Marina De Vos*[2], *Julian Padget*[2], *Ken Satoh*[3] *and Tina Balke*[4]

[1] *Institute for Security Science and Technology, Imperial College London, UK*
(*e-mail:* tingting.li@imperial.ac.uk)
[2] *Department of Computer Science, University of Bath, Bath, UK*
(*e-mail:* {mdv,jap}@cs.bath.ac.uk)
[3] *National Institute of Informatics and Sokendai, Japan*
(*e-mail:* ksatoh@nii.ac.jp)
[4] *Centre for Research in Social Simulation, University of Surrey, UK*
(*e-mail:* t.balke@surrey.ac.uk)

Declarative programming allows the expression of properties of the desired solution(s), while the computational task is delegated to a general-purpose algorithm. The freedom from explicit control is counter-balanced by the difficulty in working out what properties are missing or are incorrectly expressed, when the solutions do not meet expectations. This can be particularly problematic in the case of answer set semantics, because the absence of a key constraint/rule could make the difference between none or thousands of answer sets, rather than the intended one (or handful). The debugging task then comprises adding or deleting conditions on the right hand sides of existing rules or, more far-reaching, adding or deleting whole rules. The contribution of this paper is to show how inductive logic programming (ILP) along with examples of (un)desirable properties of answer sets can be used to revise the original program semi-automatically so that it satisfies the stated properties, in effect providing debugging-by-example for programs under answer set semantics.

KEYWORDS: Answer Set Programming, Debugging, Inductive Logic Programming

12

Learning Probabilistic Action Models from Interpretation Transitions

David Martínez[1], *Tony Ribeiro*[2], *Katsumi Inoue*[3], *Guillem Alenyà*[4] *and Carme Torras*[4]

[1] *Institut de Robotica i Informatica Industrial (CSIC-UPC)*
Llorens i Artigas 4-6, 08028 Barcelona, Spain
(*e-mail:* dmartinez@iri.upc.edu)
[2] *The Graduate University for Advanced Studies (Sokendai),*
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
(*e-mail:* tony_ribeiro@nii.ac.jp)
[3] *National Institute of Informatics,*
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
(*e-mail:* inoue@nii.ac.jp)
[4] *Institut de Robotica i Informatica Industrial (CSIC-UPC)*
Llorens i Artigas 4-6, 08028 Barcelona, Spain
(*e-mail:* {galenya,torras}@iri.upc.edu)

There have been great advances in the probabilistic planning community during recent years, and planners can now provide solutions for very complex probabilistic tasks. However, planners require to have a model that represents the dynamics of the system, and in general these models are built by hand. In this paper, we present a framework to automatically infer probabilistic models from observations of the state transitions of a dynamic system. We propose an extension of previous works that perform learning from interpretation transitions. These works consider as input a set of state transitions and build a logic program that realizes the given transition relations. Here we extend this method to learn a compact set of probabilistic planning operators that capture probabilistic dynamics. Finally, we provide experimental validation of the quality of the learned models.

KEYWORDS: Inductive Logic Programming, Learning from Interpretation Transitions, Probabilistic Planning Operators, Action Model Learning, Probabilistic Planning

Logic Programming for Cellular Automata

Marcus Völker[1] *and Katsumi Inoue*[2]

[1] *RWTH Aachen University*
Thomashofstraße 5, 52070 Aachen, Germany
(*e-mail:* marcus.voelker@rwth-aachen.de)
[2] *National Institute of Informatics*
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
(*e-mail:* inoue@nii.ac.jp)

Cellular automata can represent real-world phenomena studied in physics and biology, and have been applied to intelligent systems like artificial life and multi-agent systems. In this paper, we study a semantic preserving transformation between cellular automata and normal logic programs based on the $T_{\mathcal{P}}$-operator. In particular, a subset of normal logic programs is shown to precisely correspond to the classic grid-based cellular automata such as Conway's Game of Life. We use two automaton models: one-way bounded cellular automata, for simplicity of construction, and unbounded cellular automata, which correspond to the classic definition of grid-based cellular automata. Using this construction, some computational theorems are easily proved regarding phenomena in the configurations of one-way bounded cellular automata, and some decidability results are newly obtained on the orbits of normal logic programs.

KEYWORDS: Semantic Foundations, $T_{\mathcal{P}}$-operator, Supported Models, Cellular Automata, Decidability

14

Relating Concrete Argumentation Formalisms and Abstract Argumentation

Michael J. Maher

School of Engineering and Information Technology
University of New South Wales, Canberra
ACT 2600, Australia
(*e-mail:* michael.maher@unsw.edu.au)

Argumentation and defeasible reasoning are essentially different names for the same thing: resolving conflicting chains of reasoning in a principled way. In modern times, argumentation has been structured through Dung's introduction of abstract argumentation. Arguments are constructed from rules, and attack relations are determined; then a semantics for abstract argumentation can be applied to evaluate the arguments and determine the consequences.

However, there are very many defeasible reasoning systems that provide concrete mechanisms for drawing conclusions from defeasible rules, without formulating the problem as the construction and then evaluation of arguments. They include various systems for non-monotonic inheritance; a wide range of defeasible logics; courteous logic programs and its more recent incarnations LPDA, ASPDA and Rulelog; DeFLog; Ordered Logic; logic programming without negation as failure (LPwNF); and Defeasible Logic Programming. Surprisingly, little work has been done relating such concrete defeasible reasoning to abstract argumentation.

In this paper we identify a small fragment of defeasible rule systems that (i) defines concrete argument systems isomorphic to abstract argumentation frameworks, and (ii) produces the same conclusions in almost all of the defeasible reasoning systems mentioned above. We show a close correspondence between abstract argumentation and the fragment as interpreted by ambiguity blocking logics in the **DL** framework. As a result, those defeasible reasoning systems can mimic abstract argumentation under the grounded semantics. Similarly, some defeasible reasoning systems can mimic abstract argumentation under the stable semantics.

These results allow us to transfer complexity lower bounds from abstract argumentation to the many defeasible reasoning systems.

KEYWORDS: Defeasible Reasoning, Abstract Argumentation, Non-Monotonic Logic

CHR Exhaustive Execution - Revisited

AhmeD Elsawy[1], *Amira Zaki*[1,2], *Slim Abdennadher*[1]

[1] *German University in Cairo, Egypt*
(*e-mail:* {ahmed.el-sawy,amira.zaki,slim.abdennadher}@guc.edu.eg)
[2] *Ulm University, Germany*
(*e-mail:* amira.zaki@uni-ulm.de)

Constraint Handling Rules (CHR) is a rule-based programming language that rewrites a multi-set of constraints until a final state is reached where no more rules are applicable (Frühwirth et al. 1996). The execution is committed-choice and fired rules cannot be retracted, thus CHR does not backtrack over alternatives.

For non-confluent programs, the derived final state depends on the order of the constraints within the query, the order of the rules within the program, and the order of the constraints within the rules. Due to the committed-choice nature of CHR, it might not be possible to reach all final states for these programs. Hence the need arises to implement a mechanism to enforce a full search space exploration.

Source-to-source transformations can be used to transform CHR programs into extended ones with additional machinery. A source-to-source transformation was introduced in (Elsawy et al. 2014) to transform any given CHR program to an extended one that explores a query's full search space. There were two main problems with the proposed transformation. First, the transformed program would perform many redundant computations, which is a drawback to its performance. Secondly, the approach was presented for only one type of CHR rules known as simplification rules. The transformation requires an additional handling for other CHR rule types that was not formally investigated.

This work aims to revisit the exhaustive execution problem, by designing a more generic and efficient source-to-source transformation that enforces full-space exploration. The new exhaustive transformation will be formalized in this work by defining how it handles all types of CHR rules. An evaluation showing the gained improvement will be also shown in comparison to previous work (Elsawy et al. 2014). In addition to, this work also proposes an interesting application for exhaustive CHR that brings it closer to its declarative form.

References

Elsawy, A., Zaki, A., and Abdennadher, S. 2014. Exhaustive execution of CHR through source-to-source transformation. In *Logic-Based Program Synthesis and Transformation - 24th International Symposium, LOPSTR 2014, Canterbury, UK, September 9-11, 2014. Revised Selected Papers*, M. Proietti and H. Seki, Eds. Vol. 8981. Springer, 59–73.

Frühwirth, T., Brisset, P., and Molwitz, J.-R. 1996. Planning cordless business communication systems. *IEEE Expert: Intelligent Systems and Their Applications*, 50–55.

KEYWORDS: Declarative programming, Constraint Handling Rules, Exhaustive execution, Source-to-source transformation, Full-search space exploration

A logic-based approach to understanding lone-actor terrorism

Dalal Alrajeh[1] *and Paul Gill*[2]

Department of Computing, Imperial College London
London SW7 2RH, United Kingdom
(*e-mail:* dalal.alrajeh@imperial.ac.uk)
[2] *Department of Security and Crime Science, University College London*
London WC1H 9EZ, United Kingdom
(*e-mail:* paul.gill@ucl.ac.uk)

The need for systematic research into behavioural factors of individual terrorists has been highlighted by much recent work on terrorism. Many existing methods follow a hypothesis-testing approach in which statistical modelling and analysis of existing data is conducted to either confirm or refute a hypothesis. However, the initial construction of hypotheses is not trivial, nor is the decision upon which of the variables are to be considered relevant for the testings. It has been argued that the lack of a methodical approach to represent, analyse, interpret and infer from existing data presents a pressing challenge to the progress of lone-actor terrorism research in particular, and the terrorism field more generally.

This paper sets a new agenda for such research. We propose the use of a logic programming approach to address the shortcomings of existing methodologies in the study of lone-actor terrorism. Our method is based on transforming characteristic and behavioural codes into a logic program and applying inductive logic programming to learn hypotheses about potentially relevant factors associated with terrorist behaviour, as well as the influence of specific factors on such behaviour. This paper is an exploratory study of 111 lone-actor terrorists' target selections (civilian vs. high-value targets) and the agency of their ideological orientation in determining their target choices.

KEYWORDS: Inductive Logic Programming, Lone-Actor Terrorism, Hypothesis Generation

18

Abstract Answer Set Solvers for Cautious Reasoning

Remi Brochenin and Marco Maratea

University of Genova, Italy
(*e-mail:* {remi.brochenin,marco.maratea}@unige.it)

Abstract solvers are a recently employed method to formally analyze algorithms that earns some advantages w.r.t. traditional ways such as pseudo-code-based description. Abstract solvers proved to be a useful tool for describing, comparing and composing solving techniques in various fields such as SAT, SMT, ASP, CASP. In ASP, abstract solvers have been so far employed for describing solvers for brave reasoning tasks.

In this paper we apply, for the first time, this methodology to the analysis of ASP solvers for cautious reasoning tasks. We describe and compare the available approaches in the literature, which employ techniques for computing over- and under-approximations of the solution, the last including "coherence tests" for deciding the inclusion of a single atom in the solution, a technique borrowed from backbone computation of CNF formulas. Then, we show how to improve the current abstract solvers with new techniques, in order to design new solving algorithms.

KEYWORDS: Answer Set Programming, Abstract Solvers, Cautious Reasoning

Thread-Aware Logic Programming for Data-Driven Parallel Programs

Flavio Cruz[1,2]*, Ricardo Rocha*[2]*, Seth Copen Goldstein*[1]

[1] *Carnegie Mellon University, Pittsburgh, PA 15213*
(*e-mail:* {fmfernan,seth}@cs.cmu.edu)
[2] *CRACS & INESC TEC and Faculty of Sciences, University Of Porto*
Rua do Campo Alegre, 1021/1055, 4169-007 Porto, Portugal
(*e-mail:* {flavioc,ricroc}@dcc.fc.up.pt)

Declarative programming in the style of functional and logic programming has been hailed as an alternative parallel programming style where computer programs are automatically parallelized without programmer control. Although this approach removes many pitfalls of explicit parallel programming, it hides important information about the underlying parallel architecture that could be used to improve the scalability and efficiency of programs. In this paper, we present a novel programming model that allows the programmer to reason about thread state in data-driven declarative programs. This abstraction has been implemented on top of Linear Meld, a linear logic programming language that is designed for writing graph-based programs. We present several programs that show the flavor of our new programming model, including graph algorithms and a machine learning algorithm. Our goal is to show that it is possible to take advantage of architectural details without losing the key advantages of logic programming.

KEYWORDS: Parallel Programming, Declarative Programming, Coordination

Towards a Generic Interface to Integrate CLP and Tabled Execution (Abstract)

Joaquín Arias[1] *and Manuel Carro*[1,2]

[1] *IMDEA Software Institute*
(*e-mail:* joaquin.arias@imdea.org)
[2] *Technical University of Madrid*
(*e-mail:* manuel.carro@{imdea.org,upm.es})

Logic programming systems featuring Constraint Logic Programming and tabled execution have been shown to increase the declarativeness and efficiency of Prolog, while at the same time making it possible to write very expressive programs. Previous implementations fully integrating both capabilities (i.e., forcing suspension, answer subsumption, etc. where it is necessary in order to avoid recomputation and terminate whenever possible) did not have a simple, well-documented, easy-to-understand interface which made it possible to integrate arbitrary CLP solvers into existing tabling systems. This clearly hinders a more widespread usage of this combination.

In our work, we examine the requirements that a constraint solver must fulfill to be easily interfaced with a tabling system. We propose a minimal set of operations which the constraint solver has to provide to the tabling engine. The operations are based in only four objects (Vars, Dom, ProjStore and Store). Vars is a list with the constrained variables of a call. Dom and ProjStore represent the projection of the constraint store corresponding to a call. Store is the representation of the constraint store of a generator and is used by the external constraint solver to reinstall it when the generator is complete. The two main operations to be provided by the solver are: (i) entailment, entail(+Vars_A, +Dom_A, +Dom_B, +$\mathsf{ProjStore}_B$), which checks if the call/answer constraint store (Vars_A and Dom_A) is entailed by the previous call/answer constraint store (Dom_B and $\mathsf{ProjStore}_B$) and (ii) projection, executed in two steps: project_domain(+Vars, -Dom), that pre-computes an object used during the entailment, and project_gen_store(+Vars, +Dom, -ProjStore) which is executed when the entailment phase fails.

We validate our design with three use cases. First we re-engineer a previously existing tabled constrain domain (difference constraints) in Ciao. This solver is implemented in C, so the arguments of the interface represent the memory address of C structures. Then we integrate Holzbauer's CLP(Q) implementation with Ciao Prolog's tabling engine. Since existing CLP(Q) predicates already provide the necessary functionality, we only need to write simple bridge predicates. Last, we implement a constraint solver over (finite) lattice that is parametrized by the lattice domain. The lattice domain defines the elements and its operations, including at least join and meet which define the partial order ($\sqsubseteq$) relation used to check entailment.

We evaluate the cost of adopting a more modular framework versus the previous

non-modular implementation of difference constraints. We present the benefits of using TCLP(Q), which gives more expressiveness and in some cases better performance that TCLP(Diff). We also implemented a simple abstract analyzer whose fix-point is reached by means of tabled execution and whose domain operations are implemented using the constraint solver over (finite) lattices, which avoids recomputation of subsumed abstractions and attains better accuracy and considerable speedups.

KEYWORDS: Constraint Logic Programming, Tabling, Prolog, Interface, Implementation

22

Markov Logc Style Weighted Rules under the Stable Model Semantics

Joohyung Lee[1]*, Yunsong Meng*[2] *and Yi Wang*[1]

[1] *School of Computing, Informatics, and Decision Systems Engineering*
Arizona State University, Tempe, AZ, USA
(*e-mail:* {joolee,ywang485}@asu.edu)
[2]*Samsung Research America, Mountain View, CA, USA*
(*e-mail:* yunsong.m@samsung.com)

We introduce the language LP^{MLN} that extends logic programs under the stable model semantics to allow weighted rules similar to the way Markov Logic considers weighted formulas. LP^{MLN} is a proper extension of the stable model semantics to enable probabilistic reasoning, providing a way to handle inconsistency in answer set programming. We also show that the recently established logical relationship between Pearl's Causal Models and answer set programs can be extended to the probabilistic setting via LP^{MLN}.

KEYWORDS: Answer Set Programming, Markov Logic Networks, Probabilistic Causal Models, Probabilistic Logic Programming

On Structural Analysis of Non-Ground Answer-Set Programs[4]

Benjamin Kiesl[1], *Peter Schuller*[2] *and Hans Tompits*[1]

[1]*Institut fur Informationssysteme,*
Arbeitsbereich Wissensbasierte Systeme 184/3,
Technische Universit at Wien
(*e-mail:* {kiesl,tompits}@kr.tuwien.ac.at)
[2]*Department of Computer Engineering,*
Faculty of Engineering, Marmara University
(*e-mail:* peter.schuller@marmara.edu.tr)

The development of answer-set programs often involves domain experts without a background in logic programming. In such situations, it would be beneficial to translate programs into a form which is easier to understand and closer to natural language. Since the structure of a program determines to a great extent how a program should be explained in a clear and comprehensible way, as a first step towards a natural-language representation of answer-set programs, in this paper, we introduce methods for analysing the structure of disjunctive non-ground answer-set programs. In particular, as most programs follow the *generate-define-test paradigm*, we introduce formal definitions to characterise the respective generate, define, and test parts of a program. Thereby, we define the *non-deterministic core* of a program, effectively determining the program's active solution-space generators, following ideas of the weakly perfect model semantics as introduced by Przymusinska and Przymusinski, and we prove that our definitions fulfil desirable properties. Moreover, we also provide an implementation of a tool, using a metaprogramming approach, which classifies the rules of a given program according to our definitions. Finally, we propose an algorithm that, based on our generate-define-test classification, computes the order in which the rules of a program should be explained when translated into natural language.

KEYWORDS: Answer-set Programming, Generate-define-test Paradigm, Program Analysis

[4] This work has been supported by the Austrian Science Fund (FWF) under project W1255-N23 and by the Scientific and Technological Research Council of Turkey (TUBITAK) under grant 114E777.

24

A logical approach to working with biological databases

Nicos Angelopoulos and Georgios Giamas

Department of Surgery and Cancer, Imperial College, London, UK
(*e-mail:* nicos.angelopoulos@gmail.com,g.giamas@imperial.ac.uk)

It has been argued before that Prolog is a strong candidate for research and code development in bioinformatics and computational biology. This position has been based on both the intrinsic strengths of Prolog and recent advances in its technologies. Here we strengthen the case for the deployment and penetration of Prolog into bioinformatics, by introducing *bio_db*, a comprehensive and extensible system for working with biological data. We focus on databases that translate between biological products and product-to-product interactions, the latter of which can be visualised as graphs. This library allows easy access to high quality data in two formats: as Prolog fact files and as SQLite databases. On-demand downloading of prepacked data files in these two formats is supported in all operating system architectures as well as reconstruction from latest data files from the curated databases. The methods used to deliver the data are transparent to the user and the data are delivered in the familiar format of Prolog facts.

KEYWORDS: Biological Databases, Bioinformatics, Gene Ontology, SQLite, Big Data

Automated Reasoning about XACML 3.0 Delegation Using Answer Set Programming

Joohyung Lee[1], *Yi Wang*[1] *and Yu Zhang*[2]

[1] *School of Computing, Informatics, and Decision Systems Engineering*
Arizona State University, Tempe, AZ, USA
(*e-mail:* {joolee,ywang485}@asu.edu)
[2] *Intel, Chandler, AZ, USA*
(*e-mail:* yzhan289@asu.edu)

XACML is an XML-based declarative access control language standardized by OASIS. Its latest version 3.0 has several new features including the concept of delegation for decentralized administration of access control. Though it is important to avoid unintended consequences of ill-designed policies, delegation makes formal analysis of XACML policies highly complicated. In this paper, we present a logic-based approach to XACML 3.0 policy analysis. We formulate XACML 3.0 in Answer Set Programming (ASP) and use ASP solvers to perform automated reasoning about XACML policies. To the best of our knowledge this is the first work that fully captures the XACML delegation model in a formal executable language.

KEYWORDS: Policy, XACML, Delegation, Answer Set Programming

Unifying Justifications and Debugging for Answer-Set Programs

Carlos Viegas Damàsio[1], [5] *João Moura*[1] *and Anastasia Analyti*[2]

[1] *NOVA LINCS, Universidade Nova de Lisboa, Portugal*
(*e-mail:* cd@fct.unl.pt,joaomoura@yahoo.com)
[2] *Institute of Computer Science, FORTH-ICS, Crete, Greece*
(*e-mail:* analyti@ics.forth.gr)

Recently, (Viegas Damàsio et al. 2013) introduced a way to construct propositional formulae encoding provenance information for logic programs. From these formulae, justifications for a given interpretation are extracted but it does not explain why such interpretation is not an answer-set (debugging). Resorting to a meta-programming transformation for debugging logic programs, (Gebser et al. 2008) does the converse. Here we unify these complementary approaches using meta-programming transformations. First, an answer-set program is constructed in order to generate every provenance propositional models for a program, both for well-founded and answer-set semantics, suggesting alternative repairs to bring about (or not) a given interpretation. In particular, we identify what changes must be made to a program in order for an interpretation to be an answer-set, thus providing the basis to relate provenance with debugging. With this meta-programming method, one does not have the need to generate the provenance propositional formulas and thus obtain debugging and justification models directly from the transformed program. This enables computing provenance answer-sets in an easy way by using AS solvers. We show that the provenance approach generalizes the debugging one, since any error has a counterpart provenance but not the other way around. Because the method we propose is based on meta-programming, we extended an existing tool (Spock) and developed a proof-of-concept (http://cptkirk.sourceforge.net) built to help computing our models.

References

Damàsio, C., Analyti, A., and Antoniou, G. 2013. Justifications for Logic Programming. In *Logic Programming and Nonmonotonic Reasoning,* P. Cabalar andT. C. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer Berlin Heidelberg, 530–542.

KEYWORDS: Answer Set Programming, Debugging, Justifications, Provenance

[5] Under grant SFRH/BD/69006/2010 from Fundação para a Ciência e Tecnologia / Ministério do Ensino e da Ciêencia.

An abductive Framework for Datalog$^{\pm}$ Ontologies

Marco Gavanelli[1], *Evelina Lamma*[2], *Fabrizio Riguzzi*[2], *Elena Bellodi*[3], *Riccardo Zese*[3] *and Giuseppe Cota*[3]

[1] *Dipartimento di Ingegneria – University of Ferrara*
[2] *Dipartimento di Matematica e Informatica – University of Ferrara*
[3] *Dipartimento di Ingegneria – University of Ferrara*
(*e-mail:* {<firstname>.<lastname>@unife.it})

Ontologies are a fundamental component of the Semantic Web since they provide a formal and machine manipulable model of a domain. Description Logics (DLs) are often the languages of choice for modeling ontologies. Great effort has been spent in identifying decidable or even tractable fragments of DLs. Conversely, for knowledge representation and reasoning, integration with rules and rule-based reasoning is crucial in the so-called Semantic Web stack vision. Datalog$^{\pm}$ is an extension of Datalog which can be used for representing lightweight ontologies, and is able to express the DL-Lite family of ontology languages, with tractable query answering under certain language restrictions.

In this work, we show that Abductive Logic Programming (ALP) is also a suitable framework for representing Datalog$^{\pm}$ ontologies, supporting query answering through an abductive proof procedure, and smoothly achieving the integration of ontologies and rule-based reasoning. In particular, we consider an Abductive Logic Programming framework named $\mathcal{S}$CIFF and derived from the IFF abductive framework, able to deal with existentially (and universally) quantified variables in rule heads, and Constraint Logic Programming constraints. Forward and backward reasoning is naturally supported in the ALP framework. We show that the $\mathcal{S}$CIFF language smoothly supports the integration of rules, expressed in a Logic Programming language, with Datalog$^{\pm}$ ontologies, mapped into $\mathcal{S}$CIFF (forward) integrity constraints.

KEYWORDS: Abductive Logic Programming, Datalog$^{\pm}$, Description Logics, Semantic Web.

28

Expressing and Supporting Efficiently Greedy Algorithms as Locally Stratified Logic Programs

C. Zaniolo

University of California, Los Angeles, USA
(*e-mail:* zaniolo@cs.ucla.edu)

The problem of expressing and supporting classical greedy algorithms in Datalog has been the focus of many significant research efforts that have produced very interesting solutions for particular algorithms. But we still lack a general treatment that characterizes the relationship of greedy algorithms to non-monotonic theories and leads to asymptotically optimal implementations. In this paper, we propose a general solution to this problem. Our approach begins by identifying a class of locally stratified programs that subsumes XY-stratified programs and is formally characterized using the Datalog_{1S} representation of numbers. Then, we propose a simple specialization of the iterated fixpoint procedure that computes efficiently the perfect model for these programs, achieving optimal asymptotic complexities for well-known greedy algorithms.This makes possible their efficient support in Datalog systems.

KEYWORDS: Horn Clauses, Datalog, Aggregates, Greedy Algorithms