Association for Logic Programming Newsletter

Vol. 21, No. 1
February/March, 2008

Contents

Vol. 21 n. 1, February/March 2008

FEATURED ARTICLES

VOL. 21 NO. 1
FEBRUARY/MARCH 2008

International Conference on Logic Programming 2008!

A. Dovier, M. Garcia de la Banda, E. Pontelli
University of Udine, Italy
Monash University, Australia
New Mexico State University, USA

Editor: Enrico Pontelli

Download: POSTER
URL: http://iclp08.dimi.uniud.it/
Location: Udine, Italy
Date: December 9-13, 2008

CONFERENCE SCOPE
Since the first conference held in Marseilles in 1982, ICLP has been the premier international conference for presenting research in logic programming. Contributions (papers, position papers, and posters) are sought in all areas of logic programming including but not restricted to:

- Theory: Semantic Foundations, Formalisms, Nonmonotonic Reasoning, Knowledge Representation.
- Implementation: Compilation, Memory Management, Virtual Machines, Parallelism.
- Environments: Program Analysis, Program Transformation, Validation and Verification, Debugging, Profiling, Integration.
- Language Issues: Extensions, Integration with Other Paradigms, Concurrency, Modularity, Objects, Coordination, Mobility, Higher Order, Types, Modes, Programming Techniques.
- Related Paradigms: Abductive Logic Programming, Inductive Logic Programming, Constraint Logic Programming, Answer-Set Programming.
- Applications: Databases, Data Integration and Federation, Software Engineering, Natural Language Processing, Web and Semantic Web, Agents, Artificial Intelligence, Bioinformatics

The three broad categories for submissions are:

1. Technical papers, providing novel research contributions, innovative perspectives on the field, and/or novel integrations across different areas;

2. Application papers, describing innovative uses of logic programming technology in real-world application domains;
3. Posters, ideal for presenting and discussing current work, not yet ready for publication, for PhD thesis summaries and research project overviews.

A separate session dedicated to the celebration of the 20th anniversary of stable model semantics will also be part of the program.
Accepted papers and posters will be allocated time for presentation during the conference. At least one author of each accepted submission is expected to register and participate in the event.

In addition to papers and posters, the technical program will include invited talks, advanced tutorials, specialized sessions, workshops, and a Doctoral Student Consortium. Details, as they become available will be posted at:

http://iclp08.dimi.uniud.it

PAPERS AND POSTERS
Papers and posters must describe original, previously unpublished research, and must not be simultaneously submitted for publication elsewhere. Emphasis will be placed on the novelty and innovative nature of the results (even if not completely polished and refined). All submissions will be peer-reviewed by an international panel. Submissions MUST contain substantial original, unpublished material. All submissions must be written in English. Technical papers and application papers must not exceed 15 pages in the Springer LNCS format (see http://www.springeronline.com/lncs/)
The limit for posters is 5 pages in the same format.
The primary means of submission will be electronic, through the Easychair submission system. The submission page is available at

http://www.easychair.org/conferences/?conf=ICLP08

PUBLICATION
The proceedings of the conference will be published by Springer-Verlag in the LNCS series. All accepted papers and posters will be included in the proceedings.

WORKSHOPS
The ICLP'08 program will include several workshops. They are perhaps the best place for the presentation of preliminary work, novel ideas, and new open problems to a more focused and specialized audience. Workshops also provide a venue for presenting specialised topics and opportunities for intensive discussions and project collaboration in any areas related to logic programming, including cross-disciplinary areas.

DOCTORAL CONSORTIUM
The Doctoral Consortium (DC) on Logic Programming is the 4th Doctoral consortium to provide doctoral students with the opportunity to present and discuss their research directions, and to obtain feedback from both peers and word-renown experts in the field. The DC will also offer invited speakers and panel discussions. Accepted participants will receive partial financial support to attend the event and the main conference. The best paper and presentation from the DC will be given the

opportunity to present in special session of the main ICLP conference.

CELEBRATING 20 YEARS OF STABLE MODEL SEMANTICS

The year 2008 marks the 20th anniversary of the publication that introduced the stable model semantics for logic programs with negation. The paper titled "The stable semantics for logic programs" by Michael Gelfond and Vladimir Lifschitz was presented at ICLP-1988. It was a momentous event that gave rise to a vibrant subfield of logic programming known now as the answer-set programming. Its distinguishing aspects are close connections to the fields of knowledge representation, satisfiability and constraint satisfaction, ever faster computational tools, and a growing list of successful applications.

To celebrate the stable-model semantics, there will be a special session at ICLP 2008 dedicated to answer-set programming. The session will feature talks by Michael Gelfond and Vladimir Lifschitz. as well as by other major contributions to the field, presenting personal perspectives on the stable-model semantics, its impact and its future. There will be a panel discussion, and regular accepted ICLP papers falling into the answer-set programming area will complete the program.

CONFERENCE VENUE

The conference will be held in the city of Udine, the capital of the historical region of Friuli, Italy. Located between the Adriatic sea and the Alps, close to Venice, Austria and Slovenia, Udine is a city of Roman origins, funded by Emperor Otto in 983. Rich of historical sites, Udine is also famous for its commercial and shopping opportunities and its outstanding wine and culinary traditions.

SUPPORT SPONSORING AND AWARDS

The conference is sponsored by the Association for Logic Programming (ALP). The ALP has funds to assist financially disadvantaged participants. The ALP is planning to sponsor two awards for ICLP 2008: for the best technical paper and for the best student paper.

IMPORTANT DATES

	PAPERS	POSTERS
Abstract Submissions	June 2nd	n/a
Submission Deadline	June 9th	August 15th
Notifications	August 1st	September 1st
Camera-ready	September 15th	September 15th
Doctoral Consortium	TBA	
Workshop Proposals	June 2nd	
Early-bird Registrations	TBA	

ICLP'2008 ORGANIZATION

General Chair: Agostino Dovier (University of Udine)

Program Co-Chairs:
Maria Garcia de la Banda (Monash University)

Enrico Pontelli (New Mexico State University)

Workshop Chair: Tran Cao Son (New Mexico State University)

Doctoral Student Consortium:
David Warren (SUNY Stony Brook)
Tom Schrijvers (K.U.Leuven)

Publicity Co-Chairs:
Marcello Balduccini (Kodak Research Labs)
Alessandro Dal Palu' (University of Parma)

Programming Competition Chair: Bart Demoen (K.U.Leuven)

20 Years of Stable Models:
Mirek Truszczynski (University of Kentucky)
Andrea Formisano (University of Perugia)

Program Committee:
Salvador Abreu
Sergio Antoy
Pedro Barahona
Chitta Baral
Gerhard Brewka
Manuel Carro
Michael Codish
Alessandro Dal Palu'
Bart Demoen
Agostino Dovier
John Gallagher
Michael Gelfond
Carmen Gervet
Gopal Gupta
Manuel Hermenegildo
Andy King
Michael Maher
Juan Moreno Navarro
Alberto Pettorossi
Brigitte Pientka
Gianfranco Rossi
Fariba Sadri
Vitor Santos Costa
Tran Cao Son
Paolo Torroni
Frank Valencia
Mark Wallace

Web Master:
Raffaele Cipriano

Local Arrangements Committee:
Alberto Casagrande

Elisabetta De Maria
Luca Di Gaspero
Carla Piazza

2008 Summer School in Logic Programming and Computational Logic

Inna Pivkina and Enrico Pontelli
New Mexico State University
USA

Editor: Enrico Pontelli

URL: http://www.cs.nmsu.edu/~ipivkina/compulog.htm
Date: July 24-27, 2008
Location: Las Cruces, New Mexico, USA

The third international summer school in Logic Programming and Computation Logic will be held on the campus of New Mexico State University in beautiful Las Cruces, New Mexico.
The summer school is intended for graduate students, post-doctoral students, young researchers, and programmers interested in constraints, logic programming, computational logic and their applications. The lectures will be given by internationally renowned researchers who have made significant contributions to the advancement of these disciplines. The summer school is a good opportunity for quickly acquiring background knowledge on important areas of computational logic. The summer school is especially directed to Ph.D. students who are just about to start research, post-doctoral students interested in entering a new area of research, and young researchers (e.g., assistant professors). Exceptional undergraduate students in their senior year are also encouraged to attend.

The summer school will consist of six 1/2 day tutorials on the following topics:

- Theoretical Foundations of Logic Programming [Miroslaw Truszczynski, U. of Kentucky]
- Answer Set Programming [Torsten Schaub, U. of Potsdam]
- Implementation and Execution Models for Logic Programming [Manuel Hermenegildo, Polytechnic Univ. of Madrid]
- Logic Programming and Multi-agent Systems [Francesca Toni, Imperial College]
- Foundations of Constraint and Constraint Logic Programming [Brent Venable, University of Padova]
- Foundations of Semantic Web and Computational Logic [Sheila McIlraith, University of Toronto]

Registration
Due to the limit on the number of slots available, we invite interested student to submit an application for admission to the summer school composed of the following items:

1. a one page statement of interest, explaining your research background and what

you expect to gain from the summer school
2. a short (2-page) vitae

Applications should be submitted in electronic form to:

epontell AT cs.nmsu.edu and ipivkina AT cs.nmsu.edu

All submissions will be acknowledged with an email. If you do not receive acknowledgement within 3 working days, please email Enrico Pontelli (epontell AT cs.nmsu.edu).

Student grants
The school is free of charge for all admitted applicants. We will also provide lunches to all participants for the duration of the summer school. Several different types of grants will be available to offset partially or totally the travel and lodging costs.

IMPORTANT: Thanks to the joint support of the Computing Research Association Committee on the Status of Women in Computing Research and the Coalition to Diversify Computing, we have ***several full scholarships*** (i.e., travel and lodging) for MINORITY and WOMEN applicants. Please contact the organizers for further information.

Partial grants covering lodging and meals will be provided to other selected participants who requests them. Students who wish to request a grant should contact via email Enrico Pontelli (epontell AT cs.nmsu.edu) motivating the request.

Lodging
Lodging will be available at local hotels; we will also provide a number of affordable accomodations on the NMSU campus.

Important dates
Requests for student grants: April 19, 2008;
Application for Admission: April 27, 2008;
Notification of Admission and grants: May 5th, 2008;
Summer School: July 24-27, 2008

Local Organizers
* Enrico Pontelli, New Mexico State University, USA
* Inna Pivkina, New Mexico State University, USA
* Karen Villaverde, New Mexico State University, USA
* Son Cao Tran, New Mexico State University, USA

Intelligent Systems Laboratory (ISL)

Yan Zhang
School of Computing and Mathematics
University of Western Sydney
Australia

Editor: Son Cao Tran

About the Laboratory

Intelligent Systems Lab (ISL) comprising academic staff, research fellows and postgraduate students from the School of Computing and Mathematics, the University of Western Sydney, Australia. ISL members share research interests in computational intelligence, software development and computer security. It is the objective of ISL to undertake excellent research in these areas and their fundamental common grounds.

Currently, ISL has four academic staff, three postdoc research fellows, and six Honours and postgraduate research students. Part of ISL's culture consists in the continuing links with external researchers and groups. Each year, ISL will host a number of international researchers to visit ISL and exchange research ideas with them. ISL has also remained regularly mutual visits with researchers in Germany, Hong Kong - China, and USA.

ISL Recent Research Projects

A World View Solver for Epistemic Logic Programs

The world view semantics for epistemic logic program was proposed by Gelfond in his 1994 paper *"Logic programming and reasoning with incomplete information" (Annals of Mathematics and Artificial Intelligence, 12 (1994) 98-116)*. It extends the answer set semantics for disjunctive logic programs and allows the reasoning with the agent's beliefs and knowledge.

We have implemented a prototype called Wviews that computes the world views for a given epistemic logic program. Wviews works in the following way: it takes an epistemic logic program as input, then guesses an epistemic valuation for all subjective literals occurring in the program and transform the input program to an extended disjunctiv logic program. Finally, by calling dlv, Wviews computes all answer sets of this extended disjunctive logic program and verify whether the collection of these answer sets is a world view of the input epistemic logic program. Many detailed optimization methods have been implmented in order to prune the search space.

Wviews now can be downloaded from:
http://www.scm.uws.edu.au/~yan/Wviews.html

Theory of Forgetting in Logic Programs

In this project, we consider how to forget a set of atoms in a logic program. Intuitively,

when a set of atoms is forgotten from a logic program, all atoms in the set should be eliminated from this program in some way, and other atoms related to them in the program might also be affected. We define notions of strong and weak forgettings in logic programs to capture such intuition, reveal their close connections to the notion of forgetting in classical propositional theories, and provide a precise semantic characterization for them. Based on these notions, we then develop a general framework for conflict solving in logic programs. We investigate various semantic properties and features in relation to strong and weak forgettings and conflict solving in the proposed framework. We argue that many important conflict solving problems can be represented within this framework. In particular, we show that all major logic program update approaches can be transformed into our framework, under which each approach becomes a specific conflict solving case with certain constraints.

Relevant publications are as follows:

- Y. Zhang and N.Y. Foo, *Solving logic program conflict through strong and weak forgettings.* Artificial Intelligence (AIJ). 170 (2006) 739-778.
- Y. Zhang and N.Y. Foo, *A unified framework for representing logic program updates.* In Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005), pp 707-712. AAAI Press 2005.
- Y. Zhang, N.Y. Foo and K. Wang, *Solving logic program conflicts through strong and weak forgettings.* In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), pp 627-632. Morgan Kaufmann Publishers, Inc. 2005.

Bargaining Theory and Automated Negotiation:

Bargaining is a traditional research topic in economics initiated by John Nash in 1950. We explore the theme from AI point of view by modelling reasoning behind bargaining processes. We express bargaining situations in logical form and model bargaining games with a combinatorial form of belief revision and game theory. We have proposed a logic solution to the bargaining problem, which has a sound and complete axiomatic characterization. Such an axiomatization is a combination of belief revision postulates and game theoretic axioms. We have also proposed a set of other logic-based solutions to deal with ordinal bargaining and bargaining with mixed deals. The whole framework initiates a new methodology of bargaining analysis and would help us to identify the logical reasoning behind bargaining processes

Relevant publications are as follows:

- D. Zhang, *Reasoning about bargaining situations.* In Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07), 154-159, 2007.
- Y. Jin, M. Thielscher and D. Zhang, *Mutual belief revision: semantics and computation.* In Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07), 440-445, 2007.
- D. Zhang and Y. Zhang, *A computational model of logic-based negotiation.* In Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06), 728-733, 2006.
- D. Zhang, *A logical model for Nash bargaining solution.* In Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05), 983-988, 2005.

jackaroo Trading Agent System

Trading agent design and analysis has been one of the main research interests in ISL since 2003. We have built a trading agent system, named jackaroo, which participated in the International Trading Agent Competitions (TAC) in last five years. The agent is built upon the economic model of TAC markets and initiated the approach of equilibrium analysis to strategic trading agents. Jackaroo agent has achieved significant results in previous TAC games: the 4th place in TAC-07 CAT final, the 11th in TAC-06 SCM final, the 2nd place in TAC-05 SCM qualifying, the 1st place in TAC-05 SCM qualifying and the 3rd place in TAC-03 SCM qualifying.

Relevant publications are as follows:

- M. Furuhata and D. Zhang, *Capacity allocation with competitive retailers.* In Proceedings of the Eighth International Conference on Electronic Commerce (ICEC-06), 31-37, 2006.
- D. Zhang, *Negotiation mechanism for TAC SCM component market.* In Proceedings of the 4th International Conference on Autonomous Agent and Multiagent Systems(AAMAS-05), 288-295, 2005.
- D. Zhang, K. Zhao, C-M. Liang, G. Begum, and T-H. Huang, *Strategic trading agents via market modelling,* ACM SIGecom Exchange,4(3), 46-55, 2004.

Model Update for System Modifications

Model checking is a promising technology, which has been applied for verification of many hardware and software systems. In this paper, we introduce the concept of model update towards the development of an automatic system modification tool that extends model checking functions. We define primitive update operations on the models of Computation Tree Logic (CTL) and formalize the principle of minimal change for CTL model update. These primitive update operations, together with the underlying minimal change principle, serve as the foundation for CTL model update. Essential semantic and computational characterizations are provided for our CTL model update approach. We then describe a formal algorithm that implements this approach. We also illustrate two case studies of CTL model updates for the well-known microwave oven example and the Andrew File System 1, from which we further propose a method to optimize the update results in complex system modifications.

Relevant publications are as follows:

- Y. Zhang and Y. Ding, *CTL model update for system modifications.* Journal of Artificial Intelligence Research (JAIR) 31 (2008) 113-155.
- Y. Ding and Y. Zhang, *CTL model update: Semantics, computations and implementation.* In Proceedings of the 17th Europen Conference on Artificial Intelligence (ECAI 2006), pp 362-366. IOS Press 2006.
- Y. Ding and Y. Zhang, *A case study for CTL model update.* In Proceedings of the 1st International Conference on Knowledge Systems, Engineering and Management (KSEM 2006), pp 88-101. Springer 2006.

ISL Contact

For more information, please visit ISL web site:

http://www.scm.uws.edu.au/research/isl/

Contact: Prof Yan Zhang
Email: yan AT scm.uws.edu.au
http://www.scm.uws.edu.au/~yan/

Lazy Chickens in Grids and Cubes

Paolo Baldan
Department of Pure and Applied Mathematics
University of Padova

Roberto Bruni
Computer Science Department
University of Pisa

Dr. Egghead, after a long and brilliant scientific career, retired to his birth town, where he now runs a small farm. There he raises a strange breed of chickens, called *lazy chickens*, that originated from some secret experimentation conducted in his past life, when he was working on genetic manipulation. Lazy chickens would produce very tasty eggs, but unfortunately they are characterised by a tremendous laziness that forces them to sleep all daytime!

After some experiments, Dr. Egghead discovers how to get some eggs from lazy chickens. In fact, he realises that the presence of regular active chickens, can trigger some spirit of emulation in lazy chickens which thus start being active and producting eggs! He prepares a 8×8 grid of small cages, with a single chicken per cage and finds out that whenever a lazy chicken has two adjacent (in the sense of sharing cage sides) active chickens, then it also becomes active and it starts influencing lazy neighbours as well.

Q1: *What is the least number of regular active chickens that Dr. Egghead must insert in the cages of the* 8×8 *grid to make all chickens in the grid active? More generally, what is the solution for a generic* $n \times n$ *grid?*

As the production increases and business improves, Dr. Egghead decides to assemble the cages to form a $8 \times 8 \times 8$ cube. This time, he finds out that three active neighbours are necessary to win laziness (two do not suffice anymore).

Q2: *What is the least number of regular active chickens that Dr. Egghead must insert in the cages of the* $8 \times 8 \times 8$ *cube to make all chickens in the cube active? More generally, what is the solution for a generic* $n \times n \times n$ *cube?*

Solutions

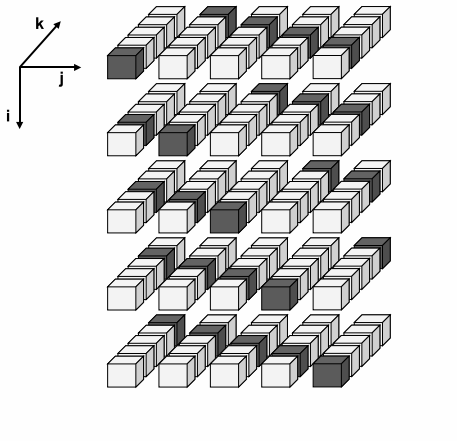
Q1: It is very easy to find a solution that involves 8 regular active chickens, e.g. placed along the main diagonal. The tricky part is to prove that 7 are not enough.

The proof of minimality can be constructed by exploiting a suitable invariant on the sides of the cages. Call a side of the cage of an active chicken *mixed* if it is not shared with another active chicken (i.e., it is shared with a lazy chicken or it is on the perimeter of the grid). We would like to reach a state of the grid where all chickens are active and thus there are exactly $8 \cdot 4 = 32$ mixed sides (i.e., all and only those on the perimeter). Now, it is immediate to check that each transformation of a lazy chicken into an active one cannot increase the number of mixed sides. But then, if we start with just 7 chickens, even if we place them in pairwise non-adjacent cages, the maximum number of mixed sides that we can have is $7 \cdot 4 = 28$.

The solution easily generalizes to a $n \times n$ grid by noticing that placing regular active chickens along the diagonal gives a solution with n chickens and that if we start with just $n - 1$ chickens then we can expect $(n - 1) \cdot 4$ mixed sides at most, as opposed to the $4 \cdot n$ arising along the perimeter when all chickens are active.

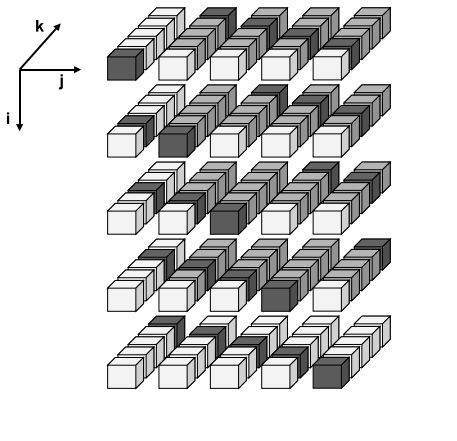
Q2: In this case the difficulties are reversed. By analogy with the grid case and exploiting a similar invariant, we can prove that at least $8^2 = 64$ regular active chickens are needed (because at the end we expect $8^2 \cdot 6$ mixed sides, those on the faces of the cube, and each active chicken can provide 6 mixed sides at most, when placed in a cage adjacent to lazy chickens only). However finding how to place them in the cube is more challenging.

This time we find convenient to define the solving scheme directly for the $n \times n \times n$ case (where n^2 cages are needed). Let us call $c_{i,j,k}$ the cage at level i from top, position j from left, and depth k from the front face of the cube (each index ranging from 0 to $n - 1$). Then, we claim that a solution with exactly n^2 regular active chickens is obtained by placing them in the cages $c_{i,j,k}$ such that $(j + k) \, mod \, n = i$. This corresponds to fill the diagonal of the bottom level and then to shift the placing by one position to the left (circularly) as we move one level up. The solution for $n = 5$ is depicted below, where the red cages (dark grey, if printed in black&white) represent active chickens.



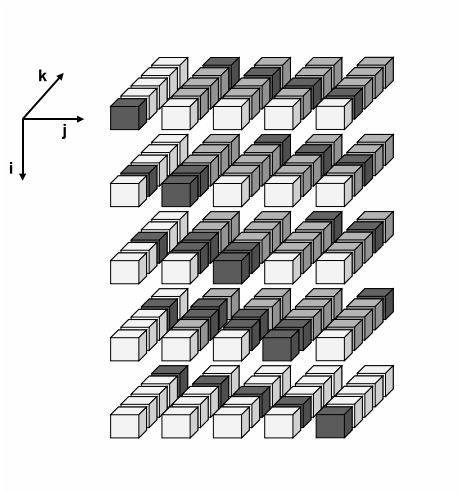
We prove that this is actually a solution with an inductive reasoning. The base case $n = 1$ is trivial. For the inductive case, assuming that the solution works for the $n \times n \times n$ cube we prove that it works for the $(n+1) \times (n+1) \times (n+1)$ cube. In order to avoid a cumbersome notation, we will be a bit informal and we will concentrate on the specific case $n = 4$.

Take the solving scheme for $n + 1 = 5$ depicted above, and consider the $4 \times 4 \times 4$ cube obtained by removing the bottom face, the front face and the leftmost face. Such sub-cube is highlighted below by using orange (medium grey, if printed in black&white) cages.

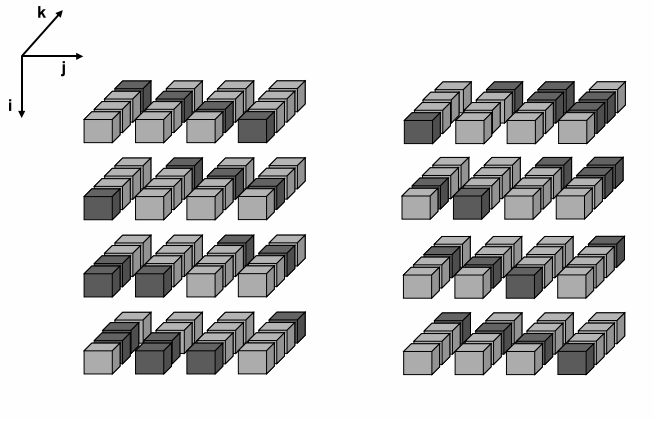


The sub-cube does not correspond to the solving scheme for $n = 4$, since it has less active chickens than needed and those present are positioned differently than expected. However, the active chickens in the removed faces can help to

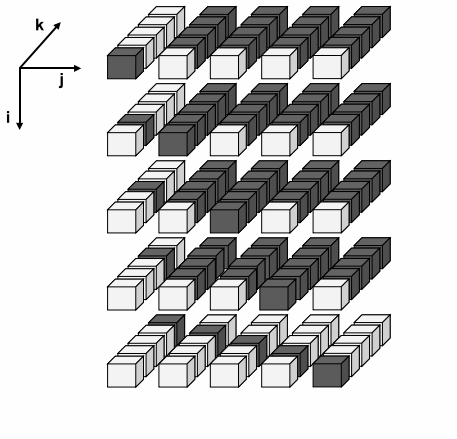
"activate" some lazy chickens in the needed positions. In fact, it is not difficult to see that we can turn some lazy chickens into active chickens in order to reach the situation depicted below



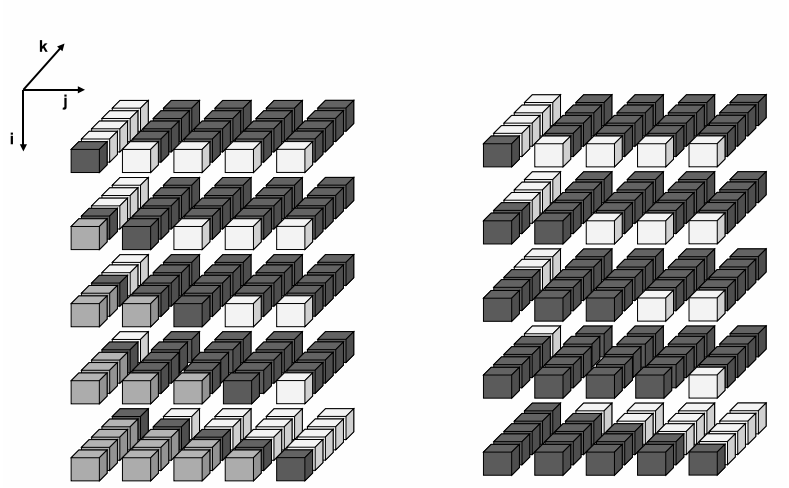
Note that the considered sub-cube, depicted in the picture below on the left, if rotated upside down, as illustrated in the right part of the figure, will contain active chickens in the cages required by the solving scheme (and some more).



Hence, by inductive hypothesis we have that all the chickens in the sub-cube can be made active, thus leading to the following situation:



Now, also the lazy chickens in cages $c_{i,j,k}$ such that $0 \leq j + k < i < n$, i.e., in orange (medium grey, if printed in black&white) cages in the picture on the left below, can be made active. We thus reach the situation illustrated in the picture on the right.



We are left to "activate" the chickens in the three triangle-shaped portions on the removed slices (the bottom level, the front face and the leftmost face): (i) $c_{n,j,k}$ with $n < j + k$, (ii) $c_{i,0,k}$ with $0 \leq i < k < n$, and (iii) $c_{i,j,0}$ with $0 \leq i \leq j < n$. In each case the problem can be easily solved by noticing that each cage of such triangles is adjacent to a face of the sub-cube where all chickens are active, so that only two adjacent active chickens on the same slice are needed. Then, since the diagonals of such slices are active by the initial placing of regular active chickens, we can immediately conclude (as in the grid case) that also the three triangle-shaped portions can be made active.

Association for Logic Programming Executive Committee Election

Enrico Pontelli
New Mexico State University
USA

Editor: Enrico Pontelli

We are pleased to announce the results of the Elections for three open positions in the Executive Committee of the Association for Logic Programming. The three positions are replacing Sandro Etalle, Gopal Gupta, and Mirek Truszczynski, who have completed their term.

The roster of nominees for this election included the following candidates:

- Piero Bonatti (University of Naples)
- Agostino Dovier (University of Udine)
- John Gallagher (Roskilde University)
- Andy King (University of Kent)
- Fangzhen Lin (Hong Kong University of Science and Technology)
- Gianfranco Rossi (University of Parma)
- Kostis Sagonas (National Technical University of Athens)
- Vitor Santos Costa (University of Porto)
- Torsten Schaub (University of Potsdam)

The ALP Secretary (David S. Warren) conducted the elections; the voting process was open to all members of ALP, as determined by the membership to the offical ALP mailing list. The tallying of the votes has been conducted with the help of Lee Naish.

The winning candidates and new members of the ALP Executive Committee are:

- Agostino Dovier
- John Gallagher
- Torsten Schaub

The ALP EC would like to thank all the candidates for their generous willingness to serve the logic programming community, and we would like to welcome the three new members. We also wish to extend a big 'THANK YOU!' to Sandro, Gopal, and Mirek for their dedication and service to the ALP over the years.

HISTORICAL PERSPECTIVES

VOL. 21 NO. 1
FEBRUARY/MARCH 2008

My Life as a Prolog Implementor

Bart Demoen
Department of Computer Science
K.U.Leuven
Belgium

Editor: Enrico Pontelli

I heard about Prolog for the first time when I was teaching IBM-370 assembler in a technical school. I was studying for a masters in informatics at that moment as well. A colleague said: "very soon now, you will learn about a language in which backtracking is build in". I thought he was joking because this did strike me as too weird. But soon afterwards, indeed, we learned about Prolog in class, and at some point the professor even gave the floor to a bearded man who tried to explain the implementation of Prolog to us: that was the first time I saw Maurice Bruynooghe. I can't remember much about that implementation, but it wasn't anything WAM-like: we speak beginning 1983.

During the summer vacation of 1983, the ministry decided that teachers would no longer be paid for overtime, so I had to quit my teaching job: another colleague at school told me about his brother Raf Venken who worked in a small company (BIM) that was looking for collaborators for a new project. He gave me the telephone number of ... Maurice: Maurice was in waiting for a permanent research job with the national FWO (kind of NSF), and in the mean time he got the manager of BIM (Michel Van den Bossche) interested in Prolog. They wrote up a project proposal which would - in collaboration with the university in Leuven - design and implement a commercial Prolog system: BIM-Prolog. At the phone Maurice asked me my qualifications and when he heard I had a masters in mathematics, I was hired. My PhD in theoretical physics was of no interest to him.

Gerda Janssens was one of the project members at the university: she had done her engineering thesis on an abstract machine for Prolog, and the idea was to use this machine as a starting point for the BIM-Prolog. She would do the implementation of the machine, I would do the compiler. We knew we still needed to do some redesign, but we were quite optimistic. Our goals were not very ambitious: beating C-Prolog in performance would have been considered a success. We looked at manuals (DEC-10 Prolog, C-Prolog, M-Prolog) and a book (Clocksin and Mellish) for learning/understanding which built-ins were needed and which we could do without. We considered some of the things we read rightout silly: how can you name a predicate tab/1 and let it write out spaces ? How stupid is it to make the space between a functor name and the next open bracket meaningful ? We did some redesign of the Prolog syntax. We were also influenced a lot by the original Marseille attitude towards Prolog. We decided on a set of built-ins and went on implementing. It was fall 1983.

In February 1984, Maurice took me to the Atlantic City ILPS. That was a wonderful experience. Also a bit awkward at some point because Maurice took me to a program committee dinner where I definitely did not belong. But most of all it was interesting, and I only realised much later who I met in person: on the way to Atlantic City we met Peter Kovacs who had been involved in the implementation of M-Prolog (we had studied the M-Prolog manuals while doing the BIM-Prolog design, especially the modules!) and Kenneth Kahn. During the conference, I was within touching distance of Alain Colmerauer who gave what might have been his first invited talk on constraint logic programming (I didn't buy it: arithmetic in which the variables are still free ... too weird for me), two giants named David Warren (an H.D. and an S.), Ehud Shapiro (who gave his talk on the Bagel), Seif Haridi, Mark Stickel and many more. It is worth looking at the proceedings of that conference, not just for the historical perspective: lots of the work presented there is really mandatory reading. The evenings in the hotel room with Maurice were also interesting: he had been given a bunch of papers to referee for some next conference (for the young: in those days you submitted 6 hard copies) and he made me read them, even though I lacked lot's of background to make sense of them.

For the BIM-KUL team, the most important ILPS paper was by Evan Tick and David H. D. Warren: Towards a Pipelined Prolog Processor. Although it didn't introduce the WAM, it was for us (Maurice and me) the first contact with the WAM - remember this is pre-internet time. After hearing the talk (Evan delivered it) Maurice and I were back in our room and Maurice asked: "What do you think ?". I answered: "Gerda will not be happy about this; we must restart from scratch". It seems that other teams went through the same experience/evolution.

So on coming back to Belgium, we restarted with the WAM as the new starting point for a faster implementation. Not that we understood the WAM at that point: some of the instructions in the Tick-Warren paper didn't make sense to us at all, so we invented our own variants, in particular related to indexing. For instance, we couldn't understand why one would create TWO choice points for one activation of a predicate, so we did away with that from the start.

While working on the first release of BIM-Prolog, a competitor emerged in the USA: Quintus-Prolog, headed by D.H.D. Warren himself, and with people like David Bowen (with whom I spend a nice evening for the first time in Atlantic City and later more at other conferences), Richard O'Keefe, Tim Lindholm, just to name some of the Quintus people I met in person, and whose names I did not forget.

At BIM, we were particularly worried about speed, the speed of nrev to be more precise: while our implementation matched Quintus-Prolog in speed for most benchmarks, we lagged far behind for nrev. BIM-Prolog had at that point already upgraded from an emulator to a native code generating system, under the impulse of two new team members: Herman Crauwels and Andre Marien. The Quintus trick to get nrev so fast was discovered by Herman by dumping the Quintus code in assembly form (somehow we had a tape with the Quintus implementation - I still have it !) and there we learned about instruction merging. All implementations do it now and it benefits not just nrev of course. We knew Quintus Prolog quite well at BIM. During the 1987 ICLP in Melbourne, where Quintus had a demo booth, I was denied access to the Quintus demo machine, because I was too good at making it crash - it was just something silly with floating point numbers.

Another player was showing at some point: SICStus Prolog, with Mats Carlsson as the main implementor. Mats was then (also) known for his LM-Prolog implementation and he gave a demo of it at the university of Brussels, probably the only place in Belgium with a Lisp Machine ever. We were very impressed. And also his later work on SICStus Prolog has always impressed me enormously: while I fled from BIM with its demanding clients back to the university - where you basically only do as you like if you play your cards right - Mats combines the development and support for a commercial system that now contains almost everything else other systems have (and it is on top of that also robust, reliable and reasonably fast), and he keeps writing interesting papers. Mats is one of my implementation heroes.

Garbage collection for Prolog was always of particular interest to me: it started (for me) with ideas by Maurice and Edwin Pittomvils (see a paper in Boston SLP 1985) and a first version of a garbage collector for BIM-Prolog which I spend about two months debugging before we decided to rewrite it from scratch - together with Alain Callebaut. Alain was also the main drive behind a full-screen debugger for BIM-Prolog, similar to dbxtool for C on SUN machines. This debugger
was far ahead of its time: it just laughed at the traditional Bird box debugging model (which I never liked to say the least - not that I think programmable debuggers are better: give me a print statement [that works even in nondet context] and I can debug anything). Garbage collection was also my first (sometimes only) involvement with some other Prolog systems: when Kostis Sagonas came to Leuven in 1996, our goal was to build a garbage collector for XSB, but before we did that, we had to understand its usefulness logic. I am so grateful to Yves Bekkers (and his colleagues) to have emphasised this concept, because the intuition that one can have about it really benefits from making it explicit. I also wanted to work on the implementation of tabling as an add-on, sort of a library that could be added to any Prolog implementation and give the benefits of the SLG-WAM: Kostis rather quickly convinced me that this is not possible, so I abandoned the idea. It is good to see that implementation groups in Madrid and Porto have made progress in this area quite recently, but important challenges still remain, and perhaps Kostis was right after all.

A very interesting development during the mid-80's was the coupling of external databases with Prolog. Unify was one such DB system and I went with Ann Mulkers (who worked like Gerda at the university on the BIM-project) to Hamburg to do such an integration - February 1986 I believe it was. It was a very exciting time. Some things we did then are now being rediscovered and published by other groups: that is good, because it means we were not too far off the right track :-)

1986 was also interesting for its ICLP in London, generally interesting, but for Prolog implementors in particular. I couldn't attend the conference, but when I saw the proceedings, I realised that we (at BIM) could have had three Prolog implementation papers there, if only we had realised that what we did in BIM-Prolog was worth writing up, instead of merely implementing it. Other papers at ICLP'86 have influenced me a lot. Chris Mellish with a paper on abstract interpretation (at Leuven we had seen a preprint of that paper and under the impulse of Maurice worked on it), and a paper by Saumya Debray. It was the first Saumya paper I ever saw, but not the last one I was impressed with. Saumya's papers show such a good mix ofscientific rigidity and pragmatism: mandatory reading for implementors I would say.

In 1987 I left BIM and went back to the university, to join the research group that Maurice was leading. Danny De Schreye also joined the group. Shortly before that,

Peter Van Roy had spend some time in Leuven and I worked with him on Prolog implementation issues.

It would be fun (for me) to go on like this, hopping from one conference to another, remembering little details from my life as an implementor while I had the opportunity to meet interesting people. Let me make some big jumps in time. BIM went bankrupt during my second visit to Paul Tarau in Moncton in 1996 (it wasn't Paul's fault!): we implemented together a novel garbage collector in BinProlog in 1995, and a Prolog to Java compiler in 1996 - just too much fun to be true. In 1997, I spend a semi-sabbatical in Melbourne with Peter Stuckey, Maria Garcia de la Banda and Kim Marriott: they seduced me to get involved in HAL, a new constraint solver programming language on top of Mercury. In the mean time, Mercury has assimilated (or has it been assimilated by ?) HAL. Also too good to be true. In
1999, I felt it was time to do my own Prolog system: not for the sake of having Yet Another Prolog (wink) but because there were too many open questions (for me) about Prolog implementation and there was no way to satisfy my curiosity with other systems - the XSB experience had somehow convinced me of that. So dProlog was born, in four incarnations (corresponding to four basic questions I had about WAM variants), one of which was implemented by Phuong-Lan Nguyen. dProlog
gave life to ilProlog (still heavily used under the name of hipP for ILP applications, maintained and further developed by Henk Vandecasteele and Remko Troncon for some time) and to hProlog which I and Phuong-Lan use regularly for experiments. hProlog was also a very good test bed for the CHR implementation by two of my PhD students, Tom Schrijvers and Jon Sneyers. It has resulted in a CHR version under SWI-Prolog, which is single-handedly maintained by Jan Wielemaker:
another hero. I have worked for a while on providing SWI-Prolog with a faster basic engine, but that failed. Since hProlog is rather speedy, I have been offered the following advice more than once: "just replace the SWI-Prolog engine with the one from hProlog". I think brain surgeons will be able to replace whole human brains sooner ...

A lot of work was done in the area of concurrent and parallel Prolog systems. There was pressure to participate in ESPRIT projects like PEPMA and ACCLAIM, and I gave in. Patrick Weemeeuw and Remco Moolenaar - two of my PhD students - worked on Parallel Garbage Collection for Aurora and on AKL. We had a ridiculously expensive 8-processor Sequent machine and it was an interesting experience. However, my attitude towards making parallel Prolog implementations is now: don't do it, it is too difficult.

I had a short involvement with GNU-Prolog: Ruben Vandeginste worked with me on garbage collection for Prolog during his PhD. He implemented a garbage collector for GNU-Prolog, I think in 2003. Unfortunately the final integration never happened. I must have contributed directly or indirectly a garbage collector to three
systems (besides BIM-Prolog and the dProlog offspring): XSB, GNU and BinProlog. None of those survived: the garbage collection level of involvement is just too intimate to last very long :-) But at least XSB and BinProlog have garbage collectors. And Daniel Diaz is now putting one in GNU-Prolog I recently heard.

What are the prospects for a Prolog implementor in 2007 ? There are several aspects to the situation. First of all it is clear that the basic implementation technology for Prolog proper is not evolving fast: in fact, the WAM is just too good. From time to time someone claims to have designed a fundamentally better machine than

the WAM, but it has always turned out that the WAM was actually superior, or the WAM is versatile enough to assimilate a particular good idea. Secondly, Prolog itself is not evolving much: having an ISO Prolog standard is part of the reason for that (and of course Part II on modules is a complete disaster). You might conclude that the prospects for a Prolog implementor look dim.

On the other hand, the application domains in which Prolog is used are evolving and become more demanding. This is a great opportunity for Prolog implementors to live out their phantasies. Just to name one field I am involved in: Inductive Logic Programming, as part of Machine Learning. In the past years I have - together with Henk, Gerda and Remko, and in cooperation with the ML team in Leuven - done work on the execution of large sets of queries on large example sets. This pushes compiler technology to the limit, requires additions to the WAM, ... just the thing one would like to be involved in when making a life as a Prolog implementor. Also, have a look at what Vitor Santos Costa (one more hero !) has been doing in Yap in the past years: quite a bit of it is inspired by the needs of ILP. For sure other
application domains do or will certainly present new challenges to the Prolog implementor.

Let me go back to the issue of Prolog as a not evolving language. There is a split - almost a schism (in the church sense, not the musical sense) - in the Prolog community (implementors mainly I think) about whether Prolog should move towards a more robust type of language where one must declare at least types, and maybe also modes. This split is very detrimental to the evolution of Prolog. Under the impulse of Zoltan Somogyi (a hero !) Mercury moved to one extreme in this respect (extreme is not meant to be judgemental) and I am not so happy with its program development model, but it is clear that the rest of the world just demands (at least) types in a programming language and for good reasons. Not having types restricts the acceptance potential of Prolog seriously. Types (and declarations) is not at all against the LP philosophy. Just complement the equation PROGRAM = LOGIC + CONTROL and get PROGRAM = LOGIC + CONTROL + TYPES. The challenge for Prolog implementors is to integrate types in the implementation in a decent way, and I am not thinking primarily about performance gains - I surprised you here, didn't I :-) BTW, I am still partly in the abstract interpretation and program analysis camp, but I believe that the equation PROGRAM = LOGIC + ANALYSIS has only imaginary solutions.

A relatively recent challenge is put to the Prolog implementors by ASP. I am using ASP in a generic way for a bunch of LP paradigms or systems based on model generation. ASP is currently the closest to what LP promised to be: you specify in a logic formalism and the implementation will find the answers to your problem. It is very much only the LOGIC part of the equations above, but that has always been the first part to get right. Still, the world needs the CONTROL part also very badly and as far as ASP goes, we are not there yet. Integrating a model generation paradigm with a goal solving paradigm has been investigated already (look at work by Enrico Pontelli for instance), but a lot more is needed. The Prolog language and its implementation assimilated constraint programming with so much grace, and also the concept of tabling. It would be a great if Prolog could also embrace ASP without sacrificing its own spirit. I consider that the most important challenge for Prolog implementors at this moment, and one which cannot be met without the help of theory (I didn't say "theoreticians" on purpose :-)

Prolog has always been our most solid stepping stone from imperative to logic

programming (and back), so it is difficult to make the next step. However, we need to move on: implementors can make it happen. Maybe only a new generation of Prolog implementors can do it.

SYSTEMS SPOTLIGHT

VOL. 21 NO. 1
FEBRUARY/MARCH 2008

Logic Programming Systems
The SWI-Prolog Environment

Jan Wielemaker
Human-Computer Studies Laboratory,
University of Amsterdam,
Kruislaan 419,
1098 VA Amsterdam,
The Netherlands,
wielemak AT science.uva.nl

Editor: Enrico Pontelli

PDF Version available HERE.

System Web Page: http://www.swi-prolog.org/

Abstract: *Development of the SWI-Prolog environment has started in 1985. Its developments was started in the context of interactive application development. More recently the development is guided by Semantic Web application development and contributions from the world wide community. In this article we will briefly introduce the SWI-Prolog community, touch the many features and outline our future plans.*
The primary aim of the Logic Programming Systems series is to provide an overview of system features. We deliberately concentrate more on the context in which the development of SWI-Prolog started, how this is currently shaped and what our plans are. A big table is more suitable for a product comparison. An outline of the community is hopefully more pleasant to read and provides more stable information about the system.

1. The SWI-Prolog history and community

We started SWI-Prolog in a EU project (KADS) where we built a workbench for Knowledge Engineering. Several partners had a background in AI and where used to Prolog. These were the days where graphical user interfaces just started to emerge, in our case SunView on SUN workstations. Anjo Anjewierden created an object-oriented connection between Prolog and SunView, called PCE [**7**]. The initial system used Quintus Prolog which, in those days,[1] could call C, but could not be called from C. This limitation severely handicapped PCE.

In the meanwhile SWI-Prolog was created as a toy Prolog system, but designed with a bi-directional C interface from the start. Just for play, we connected it to PCE and added enough built-ins to run our workbench-under-development. Enhanced functionality of PCE resulting from the bi-directional interface, a very fast compiler and the `make/0` feature to recompile modified source files quickly moved the whole consortium to use this new Prolog. As we had no concrete plans with it, we put it the source on the anonymous ftp server under a non-commercial-use license.

Somehow the sources were picked up, as it turned out mostly by Universities as the default system for teaching Prolog classes. Most likely because the system was simple, small, installed easily on most Unix systems in common use and it could be used without going through the legal and financial departments. We established a mailinglist, continued development in various projects and frequently released new versions.

PCE turned into XPCE when X11 became fashionable and the whole system was ported to Windows using Windows-NT 3.51. (X)PCE was distributed separately as a licensed system with a limited free demo for Windows. With help from Richard Stallman from the Free Software Foundation we established a transparent license schema that allowed for using the GNU readline library for editing input lines. After convincing the University we re-released the combined XPCE/SWI-Prolog system, using the LGPL for all C-code and the GPL with a statement originating from the GCC runtime libraries for the Prolog code. This statement allows using the Prolog code from the kernel and libraries alongside proprietary code. SWI-Prolog 5.0/XPCE 6.0 was released under these conditions early 2002. This proved a big step forwards[2], inviting both commercial usage and a community that contributed code in addition to bug reports and feature requests.

Commercial users sponsored the development of the SSL (Secure Socket Layer) interface, big integer and rational number support, literate programming (PlDoc) and the testing environment (PlUnit).

In 2003 Bart Demoen convinced us to add attributed variables based on the hProlog approach for dynamic attributes, after which the Leuven team would port their constraint programming tools to SWI-Prolog. Tom Schrijvers has created CHR for SWI-Prolog, Leslie de Koning ported clp(q) and clp(r). Markus Triska added several specialised constraint solvers and is currently developing a full fledged clp(fd) system. Paul Singleton developed and maintains JPL, the Java interface.

In the meanwhile internal projects started concentrating on the Semantic Web and Web services, resulting in libraries for RDF storage [**9**,**11**] and a comprehensive multi-threaded HTTP server infrastructure [**10**].

The central focus of the community is the mailinglist with approximately 600 members exchanging about 800 messages annually (2007). Second is a Bugzilla server for bug reports which functions well. We have been running a Wiki for several years to invite exchange of ideas and code. It turned out the contribution by spammers was far larger than the community contribution, forcing us to disable write access. Possibly we were too early. We are still looking for a platform that promotes exchange and improvement of reusable Prolog code in the community like Perl's CPAN[3] network.

With SWI-Prolog, we aim at providing a scalable robust and portable Prolog system to the educational, research and commercial community. We release about every two weeks or immediately after a bug has been fixed that is likely to affect many users or is a blocker for just a few users. This guarantees everyone has access to the latest bug fixes and features. We adopt the Open Source policies as outlined by Linus Torvalds, where we release new features in an early and often incomplete state, waiting for early adopters in the community to help settling the final specifications and make the library stable. This approach has worked notably well for the ODBC interface, multi-threading, unbound integer and rational number support, RDF and the HTTP services. This approach appears to work because

1. Initial development of a prototype generally takes only 10%-20% of total development time in traditional development. After this initial development the community starts to contribute with feature requests (saving specification time) and bug reports (filling the test suite).
2. Early adopters can help shaping the final system. If the developers make sure issues are fixed quickly (hours, at most a day), the users stay satisfied. Perceived missing functionality is discussed between developer and user and efficiently resolved either by extending the library or discovering how the problem can be solved appropriately using the available functionality.
3. Early adopters generally install the required software to build SWI-Prolog from source and follow the version from the SCM system (currently GIT, see below) to bypass the bi-weekly release cycle.

Since October 2007 we moved revision control from CVS to GIT.[4] GIT provides a much better web interface, is much faster and allows for distributed and offline version management. These features make it much easier for developers to cooperate or fork their own version without the need for elaborate right management.

2. SWI-Prolog by feature

2.1 Core features

SWI-Prolog's kernel is loosely based on the ZIP [**1**,**3**] virtual machine. The Prolog engine is not designed for speed, but has all commonly seen optimisations to avoid memory exhaustion in 24*7 services: last call optimisation, garbage collection and atom garbage collection. The kernel is designed to impose few limits. Atoms length, integer size and term arity is bound by memory only and atoms cover the full Unicode character set, including 0-characters.

Emerging from the need in interactive applications and network services we support for multi-threading. Multi-threading [**5**] is currently under discussion in the ISO WG17 group where the initial specification is based on the SWI-Prolog manual. The current draft is implemented in SWI-Prolog, XSB and YAP.

The primitives offered by SWI-Prolog aim at satisfying a wide user community. Built-ins and libraries inherits from different Prolog traditions: Edinburgh, ISO and Quintus/SICStus. Language constructs include mutable terms, global variables, attributed variables and coroutining based on attributed variables. The system supports

both backtrackable and non-backtrackable mutable terms and global variables. Non-backtrackable mutable data on the stacks is often used as a thread and exception-safe building block for primitives whose implementation requires data to survive backtracking, such as variations on the all-solution predicates (e.g., resource bounded `findall/3`).

Portability has always been an important asset. The current core system depends on the Ansi C99 standard for fixed-width integers and Unicode handling, POSIX threads (using pthreads-win32[5] mixed with Windows primitives for performance on Windows), the GNU GMP library for unbounded integers and rational numbers and optionally the GNU readline library for input editing. XPCE graphics depends on either X11 or the Windows GDI. The system compiles on both 32-bit and 64-bit hardware, including the 64-bit editions of Windows XP and Vista. Database connectivity depends on ODBC, using UnixODBC[6] on Unix systems. Secure Socket connections depend on OpenSSL[7]. Official releases include binaries for Linux, 32-bit and 64-bit Windows and MacOS X.

2.2 Development environments

The standard development environment is merely a set of components that provide hooks to replace parts of it by alternatives. The downside is that there is no integrated central graphics window that exposes all features. There are some alternatives that provide for a common front-end. Notably SWI-Prolog Editor[8] by Gerhard Röhner (Windows only) and an extensive GNU Emacs mode[9] by Markus Triska. These tools provide access to the native IDE components which we will briefly summarise here.
PceEmacs. PceEmacs is an Emacs clone written in XPCE/SWI-Prolog. It allows for opening the edit buffer as a Prolog stream. This feature is used to realise syntax colouring based on cross-referencing and actual parsing. The clause under the cursor is re-parsed on every keystroke. It is not coloured if it cannot be parsed, providing adequate and immediate feedback on syntax errors. Colouring of goals is based on cross-referencing using the same library as `gxref/0` discussed below and distinguishes between undefined, local, imported, and dynamic predicates while providing a context sensitive menu. Colouring of variables signals singletons and highlights variables sharing with the one under the cursor.

- **Graphical tracer.** [6] The graphical tracer is a source-level tracer that exploits PceEmacs to show the source and current location. Additional windows provide variable bindings and a stack-view that includes choice points. Notably the latter proves efficient for fixing determinism problems.
- **Graphical cross referencer.** The `gxref/0` predicate analyses the currently loaded program, providing a graphical overview of file dependencies and a per-file dependency view including not-called predicates. The underlying cross-referencer can be called as a Prolog library. It provides hooks that allow for resolving complicated meta-calling.
- **Profiler.** [6] The execution profiler samples the execution, creating the actual call-tree and how time and calls propagate along the edges. A graphical front-end allows navigating this tree.
- **PlDoc.** [8] PlDoc provides for literate programming support using Wiki-style comments. Feedback is immediate through the built-in web-server. PlDoc

integrates the user documentation with the system libraries by processing the HTML system documentation. It also provides a LaTeX backend.

- **PlUnit.** PlUnit is a test environment. It provides management when tests are loaded and executed and various levels of feedback on the progress of the tests. Tests themselves look like an ordinary clause, where the head defines properties of the test such as whether it succeeds of fails, which bindings it must produce, which errors it raises, whether or not it is deterministic as well as optional setup and cleanup handlers. The clause body contains the code that is tested. PlUnit is now used for a large part of the system tests. PlUnit runs on SWI-Prolog and SICStus Prolog.

2.3 Libraries

SWI-Prolog aims at programming the the large. Its libraries include many of the de-facto libraries such as lists, ordered sets, association lists, graphs, etc. Above all though, it contains many libraries to deal with external data formats and protocols. Many of these are wrappers around C libraries, some are native Prolog. We will just reference supported protocols and formats using their acronyms. Full information is easily found on the web and SWI-Prolog manuals. SWI-Prolog provides bundled support for C, C++. CGI, Deflate (zlib), Graphics (XPCE connects to X11 or Windows), HTTP (client and server), Java, JSON, MIME, ODBC, RDF (XML and Turtle) [**11**], SGML/HTML/XML (ported to XSB), SSL, base64, sockets (TCP and UDP) and www-form-encoded.

3. SWI-Prolog plans

Project driven development currently concentrates on enhancing the Semantic Web and general web programming support. We think this is a fruitful domain for Prolog because of its natural handling of the RDF triple model. As web services get more rule based components, the use of Prolog for web programming becomes more important. Using AJAX[10] technology and a Prolog embedded HTTP server is becoming an interesting alternative for interactive Prolog applications.

A pilot project proved the need for and feasibility of restructuring the virtual machine for better modularity and faster execution. This project should also add tabling [**4**] to SWI-Prolog.

At the ICPL-07 (Porto), we discussed the options to merge with YAP[11] Prolog. The current plan is to make it step by step easier to write code that is portable between the two dialects. The first step has been implemented and established standards for conditional compilation and detecting of dialect and system features. The next step is largely agreed and extends the module system with re-export and import by another name to make it easier to define new modules in terms of old modules, which we will use easily emulate our libraries and establish a common library structure. We also plan to arrive at a common foreign language interface. We
discuss additions we think are of mutual interest so they are shared upfront and we tackle incompatibility issues on incident basis.

4. Conclusions

We described the history, community, feature and plans around SWI-Prolog. SWI-Prolog was developed for in-house prototype development on behalf of our research. It grew big in education. Especially after a re-launch as integrated system under a clear Open Source license we have seen increasing contributions from the community and interest from research and commercial users. It aims at supporting a large part of the logic programming community with a portable, robust and feature rich implementation of the Prolog language and achieves these goals by maintaining a supportive community and short release cycle. We aim at improving the support by teaming up with YAP.

Acknowledgements

SWI-Prolog is a community. The text above already mentions the contributors of important parts of the system. Bob Wielinga is not only an active user, but also ensured projects in which SWI-Prolog could be deployed and developed. Paulo Moura builds the MacOS Intel binaries, provides Logtalk [**2**] and promotes standards in the Prolog community including the multi-threading API. Richard O'Keefe is invaluable in answering Prolog design issues on the mailinglist and providing background information for motivating built-ins and library primitives.[12] Bart Demoen is invaluable explaining implementation tricks, especially those that have not been published. Ulrich Neumerkel recently performed exhaustive tests, that helped fixing many obscure bugs.

References

[1] D.L. Bowen, L.M. Byrd, and WF. Clocksin. A portable Prolog compiler. In L.M. Pereira, editor, *Proceedings of the Logic Programming Workshop* 1983, Lisbon, Portugal, 1983. Universidade nova de Lisboa.

[2] Paulo Moura. *Logtalk - Design of an Object-Oriented Logic Programming Language*. PhD thesis, Department of Informatics, University of Beira Interior,
Portugal, September 2003.

[3] Ulrich Neumerkel. The binary wam, a simplified prolog engine. Technical report, Technische Universität Wien, 1993.
http://www.complang.tuwien.ac.at/ulrich/papers/PDF/binwam-nov93.pdf.

[4] I.V. Ramakrishnan, Prasad Rao, Konstantinos Sagonas, Terrance Swift, and David S. Warren. Efficient tabling mechanisms for logic programs. In Leon Sterling, editor, *Proceedings of the 12th International Conference on Logic Programming*, pages 697-714, Cambridge, june 1995. MIT Press.

[5] Jan Wielemaker. Native preemptive threads in SWI-Prolog. In Catuscia Palamidessi, editor, *Practical Aspects of Declarative Languages*, pages 331-345, Berlin, Germany, december 2003. Springer Verlag. LNCS 2916.

[6] Jan Wielemaker. An overview of the SWI-Prolog programming environment. In Fred Mesnard and Alexander Serebenik, editors, *Proceedings of the 13th International Workshop on Logic Programming Environments*, pages 1-16, Heverlee, Belgium, december 2003. Katholieke Universiteit Leuven. CW 371.

[7] Jan Wielemaker and Anjo Anjewierden. An architecture for making object-oriented systems available from Prolog. In Alexandre Tessier, editor, *Computer Science, abstract*, 2002. http://lanl.arxiv.org/abs/cs.SE/0207053.

[8] Jan Wielemaker and Anjo Anjewierden. PlDoc: Wiki style literate programming for Prolog. In Patricia Hill and Wim Vanhoof, editors, *Proceedings of the 17th Workshop on Logic-Based methods in Programming Environments*, pages16-30, 2007.

[9] Jan Wielemaker, Michiel Hildebrand, and Jacco van Ossenbruggen. Using Prolog as the fundament for applications on the semantic web. In S.Heymans, A. Polleres, E. Ruckhaus, D. Pearce, and G. Gupta, editors, *Proceedings of the 2nd Workshop on Applicatiions of Logic Programming and to the web, Semantic Web and Semantic Web Services*, pages 84-98, 2007.

[10] Jan Wielemaker, Zhisheng Huang, and Lourens van der Mey. SWI-Prolog and the Web. Accepted for publication in TPLP, HCS, University of Amsterdam, 2006.

[11] Jan Wielemaker, Guus Schreiber, and Bob Wielinga. Prolog-based infrastructure for RDF: performance and scalability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - Proceedings ISWC'03, Sanibel Island, Florida*, pages 644-658, Berlin, Germany, october 2003. Springer Verlag. LNCS 2870.

---

[1]Quintus, as most other todays Prolog systems offer a good bidirectional C interface.
[2]http://www.swi-prolog.org/statistics.html
[3]http://www.cpan.org/
[4]http://git.or.cz/
[5]http://sourceware.org/pthreads-win32/
[6]http://www.unixodbc.org/
[7]http://www.openssl.org/
[8]http://lernen.bildung.hessen.de/informatik/swiprolog/indexe.htm
[9]http://stud4.tuwien.ac.at/\string~e0225855/ediprolog/ediprolog.html
[10]http://en.wikipedia.org/wiki/AJAX
[11]http://www.ncc.up.pt/~vsc/Yap/
[12]http://www.cs.otago.ac.nz/staffpriv/ok/pllib.htm

REGULAR COLUMNS

VOL. 21 NO. 1
FEBRUARY/MARCH 2008

Dissertations in Logic Programming

An Open Invitation:
The column on dissertations in Logic Programming is aimed at widely publicizing recent dissertations (and even MS Theses) deadling with topics relevant to Logic Programming. This is an outstanding opportunity to

- Shine a spotlight on the new generation of logic programmers
- Demonstrate that the field of logic programming is alive with new "blood"
- Provide to our graduating students with an additional opportunity to get themselves known to the logic programming community at large
- Perhaps facilitate the creation of contacts with potential employers

Please, if you are a student who is about to complete a Thesis or Dissertation in logic programming, if you are a faculty member who is advising a student completing a Thesis in logic programming, if you have a friend/colleague/relative/... who meets such requirements, please send me a message and help me advertising these fantastic achievements.

Enrico

Integrating ASP and CLP Systems: Computing Answer Sets from Partially Ground Programs

Veena S. Mellarkod
Texas Tech University

Answer set programming (ASP) has emerged as a declarative paradigm for knowledge representation and reasoning. In this approach, a logic program is used to represent the knowledge of the domain and various tasks are reduced to computing answer sets of this program. ASP languages A-Prolog and CR-Prolog have been proven as powerful tools for constructing complex reasoning systems.
Constraint logic programming (CLP) emerged as an alternate paradigm through the fusion of logic programming and constraint solving. A CLP solver thus integrates resolution techniques from logic programming and constraint solving techniques from constraint satisfaction. While ASP is expressive for knowledge representation and reasoning, CLP solvers are efficient reasoning with numerical constraints.

Every state-of-the-art ASP solver computes answer sets of programs from their ground equivalents. Though these systems solve large industrial problems, the ground programs become huge and unmanageable. This is especially true when programs contain variables that range over large numerical domains; huge memory requirements eventually force the solvers to fail. The goal of this research is to address this issue by integrating different types of reasoning techniques to compute answer sets of programs.

First, we investigate the integration of answer set reasoning, a form of abduction, and constraint solving techniques. We design a collection of languages, V(C), parameterized over a class C of constraints. An instance AC0 from this family is studied as a knowledge representation tool. An algorithm to compute answer sets of AC0 programs is developed. An AC0 solver is built that computes answer sets from partially ground programs. The use of this language and the efficiency off the solver are demonstrated.
We extend our investigation to develop methods to include resolution techniques. We design a collection of languages AC(C) parameterized over a class C of constraints. We develop an algorithm to compute answer sets of AC(C) programs from their partial ground instances by integrating the four reasoning techniques and prove correctness. A solver is built to compute answer sets for a class of AC(C) programs.

Our work is a significant step to declaratively solve problems that cannot be solved by pure ASP or CLP solvers. The solvers built are the first to tightly integrate different reasoning techniques to compute answer sets from partial ground programs.

Efficient SAT-based Answer Set Solver

Zhijun Lin
Technical Tech University

Recent research shows that SAT (propositional satisfiability) techniques can be employed to build ecient systems to compute answer sets for logic programs. ASSAT and CMODELS are two well-known such systems that work on normal logic programs. They find an answer set from the full models for the completion of the input program, which is (iteratively) augmented with loop formulas. Making use of the fact that, for non-tight programs, during the model generation, a partial assignment may be extensible to a full model but may not grow into any answer set, we propose to add answer set extensibility checking on partial assignments.

Furthermore, given a partial assignment, we identify a class of loop formulas that are "active" on the assignment. These \active" formulas can be used to prune the search space. We also provide an efficient method to generate these formulas. These ideas can be implemented with a moderate modification on SAT solvers. We have developed a new answer set solver SAG on top of the SAT solver MCHAFF. Empirical studies on well-known benchmarks show that in most cases it is faster than the state-of-the-art answer set solvers, often by an order of magnitude. For disjunctive logic programs, the existing SAT-based solvers translate them into propositional formulas based on a complex completion definition, and then make use of loop formulas and SAT solvers to find answer set. In this paper we present a new approach that allows the translation of a program into a formula that is weaker but less complex than the completion. It performs answer set checking on partial assignments.

In case a partial assignment is inextensible to an answer set, we use support formulas, which is a generalization of loop formula, to prevent the repetition of the same mistake. Empirical studies on disjunctive logic programs con□rm the performance

advantage of the new approach.

ALP Executive Committee Report

Editor: Maria Garcia de la Banda
Secretary: David S. Warren

The ALP Executive Committee had its annual meeting during ICLP'07, in Portugal with a very full agenda that included items such the upcoming ALP EC elections, the status of the ALP domain name and website, the organisation of the current and next ICLP conferences, and the creation of a Summer School.

The main points discussed/agreed during the meeting can be summarised as follows:

- The ALP EC Elections will be held this fall with members Sandro Etalle, Gopal Gupta, and Mirek Trusczcynski stepping down. A small subcommittee was set to organise the elections
- The committee welcomed the news that delay from acceptance to appearance at the Journal of TPLP is decreasing. However, the journal is receiving fewer submissions, and thus the committed is asked to encourage more submissions. It was noted that papers published in TPLP are also permitted to appear on authors' homepages. The committee further discussed the role of Journal vs. Conference publications in the Computer Science community and the issues raised when comparing CS publication records with those of other fields. It was decided to explore the possibility of publishing the ICLP proceedings in a series, thus giving it more of the prestige of a journal in the eyes of others.
- Moshe Vardi is continuing his efforts to explore the possibility of a special interest group in the ACM on logic, SigLog.
- There are still difficulties in getting articles for the newsletter. The committee would like to receive suggestions for people who could write historical perspectives, to continue that ALP series.
- The committee welcomed the proposal of a Logic Programming Summer School by Enrico Pontelli to be held in Las Cruces, New Mexico. **[PLEASE see the article in this issue of the newsletter]**
- The committee is glad to report that the state of Logic Programming in Wikipedia has significantly improved. However, more work is needed. In particular it needs biographies (but not self-biographies.)
- The Chair reported that the ALP finally has a domain name (**http://www.logicprogramming.org**) but the website still needs significant work to bring it up to date. Pat Hill volunteered to look into how the ALP website can be improved and maintained. She suggested a mechanism that would get the Newsletter content onto ALP Website automatically.
- Regarding the status of ICLP conferences:
    - ICLP07 has been very successful. At the time of the meeting, there were 162 attendees including those for the Workshops and, thanks to help of several sponsors, it will make a nice profit. While the number of submitted papers to the Doctoral Consortium was smaller than hoped for, it is clear

that the experience at was good. It was thus decided to continue it and to further improve its advertising.

- The ICLP'08 Co-PC chairs circulated a preliminary call for papers and rised the issue of the exact definition of a "student paper". After a brief discussion, it was recommended for a paper to be considered a Student Paper if a student is the first author and that other authors certify that it is indeed primarily the students work.
- The committee would like to co-locate ICLP'09 with CP, IJCAI or AAAI, etc. However, it was decided to delay any decisions until the new EC members are elected, and the PC chair for ICLP09 is chosen.
- It was agreed for ICLP'10 to be held with FLOC10, which will be in Edinburgh.

Community News

List of News:

Systems Announcements

- **Datalog Educational System 1.5.0**
- **Abella Theorem Prover**

Books Announcements

- **Constraint Programming Letters - Second Volume**

Other Announcements

- **CFP: Special Issue on Automated Deduction**
- **Prolog BLOG**
- **CHR Workshop - Call for Submissions**
- **PhD Studentships at Kent University**
- **European Agent Systems Summer School**
- **Research Position in Bioinformatics**
- **SUMO ATP Challenge**
- **E.W.Beth Dissertation Prize**
- **European Masters Program in Computational Logic**
- **Ph.D. and PostDoc Position at the University of Leipzig**

Software Announcement
Datalog Educational System 1.5.0

Communicated by Fernando Saenz Perez

URL: http://des.sourceforge.net/

Version 1.5.0 of DES adds to previous version (1.4.0):

Enhancements:

- A more fine-grained debugging as long as individual clauses can be inspected

- Warning and error messages provided for:
  - Undefined predicates which are called by rules each time the database is changed
  - Unsafe rules
  - Execution exceptions known at compile-time
- Exception messages provided for:
  - Execution exceptions unknown at compile-time
- Rule transformation for allowing computation of safe rules which may raise run-time exceptions due to built-ins
- Rejection of unsafe or uncomputable queries, views and autoviews
- Catching of instantiation errors
- Rule source annotated for debugging and informative errors, i.e., file and lines in the program (if consulted) or assertion time (if manually asserted)
- Elapsed time display
- New basic, simpler (although less efficient than the already implemented) algorithm for computing stratified negation, following [SD91]
- Fresh variables are given new variable names instead of numbers
- New commands:
  - /negation Displays the selected algorithm for solving negation
  - /negation Algorithm Sets the required Algorithm for solving negation (strata or et_not)
  - /timing Displays whether elapsed time display is enabled
  - /timing Switch Enables or disables elapsed time display (on or off, resp.)
  - /safe Displays whether program transformation is enabled
  - /safe Switch Enables or disables program transformation (on or off, resp.)
- Changed commands:
  - /verbose Displays whether verbose output is enabled
  - /verbose Switch Enables or disables verbose output messages (on or off, resp.)
- Deprecated commands:
  - /noverbose
- Slight modifications on existing commands:
  - /debug Goal Level The inspection level can be set with the second optional argument with p for predicate level and c for clause level
  - /status Now, it also displays the selected algorithm for negation and whether program transformation is enabled
  - /version For matching the "standard" display
- New examples added to the directory examples
- The Prolog database corresponding to the Datalog loaded programs has been discarded, therefore using only one representation for them
- Revised and upgraded user's manual

Changes:

- Inequality built-ins cause an error and stops execution whenever they are computed with any non-ground argument (formerly, they silently failed)

Fixed bugs:

- The Linux version did not work. Now, it has been fixed and tested on Ubuntu 6.10, Kubuntu 7.04 (Feisty), and Mandriva Linux 2007 Spring
- The parser did not detect that the argument of not could be a variable

- Name clashes when loading programs and asserting rules are avoided

Special Issue Announcement
Journal of Symbolic Computation
Automated Deduction: Decidability, Complexity, Tractability

Communicated by V. Sofronie-Stokkermans

URL: http://www.mpi-inf.mpg.de/~sofronie/addct-special-issue.html

This special issue is devoted to the scope of the workshop ADDCT'07: Automated Deduction: Decidability, Complexity, Tractability, which took place in Bremen (Germany) on July 2007. Topics of interest include
(but are not restricted to):

- Decidability:
    - decision procedures based on logical calculi such as: resolution, rewriting, tableaux, sequent calculi, or natural deduction
    - decidability in combinations of logical theories
    - specialized decision procedures
- Complexity:
    - complexity analysis for fragments of first- (or higher) order logic
    - complexity analysis for combinations of logical theories (including parameterized complexity results)
- Tractability (in logic, automated reasoning, algebra, ...)
- Application domains for which complexity issues are essential (verification, security, databases, ontologies, ...)

Submission procedure

Submission to this special issue is completely open. We expect original articles (typically 15-30 pages; submission of larger papers will be evaluated depending on editorial constraints) that present high-quality contributions that have not been previously published in an archival venue and that must not be simultaneously submitted for publication elsewhere.

Submissions must comply with JSC's author guidelines. They must be written in English and should be prepared in LaTeX using the "Elsevier Article Class (elsart.cls)" with "JSC add-on style (yjsco.sty)" and "Harvard style references (elsart-harv.bst)". The

package "JSC LaTex" (that contains all the necessary style files and a template) can be obtained from http://www4.ncsu.edu/~hong/jsc/JSC_LaTex_2007_Mar_12.zip

The introduction of the paper MUST explicitly address the following questions in succinct and informal manner:

- What is the problem?
- Why is the problem important?
- What has so far been done on the problem?
- What is the contribution of the paper on the problem?
- Is the contribution original? Explain why.
- Is the contribution non-trivial? Explain why.

Submission to this special issue are hereby encouraged via the EasyChair submission system (http://www.easychair.org/conferences/?conf=addctjsc2008)..
The deadline for submissions is April 6th, 2008.

Guest editors:

Silvio Ghilardi (U. Milano)
Ulrike Sattler (U. Manchester)
Viorica Sofronie-Stokkermans (MPI,Saarbruecken)
Ashish Tiwari (Menlo Park)

Contact

For further informations please send an e-mail to Viorica Sofronie-Stokkermans (e-mail sofronie@mpi-inf.mpg.de)

General Announcement
Prolog BLOG

Communicated by Tom Schrijvers

URL: http://www.cs.kuleuven.be/~toms/PlanetProlog/

I was wondering whether there are more Prolog and Logic Programming bloggers out there? I started Planet Prolog, a blog aggregator for Prolog blogs:

http://www.cs.kuleuven.be/~toms/PlanetProlog/

If you also blog about Prolog or Logic Programming related stuff, and you'd like to join Planet Prolog, please let me know and send me your name and the URI of your blog's feed.

Thanks,

Tom

Workshop Announcement
Fifth Workshop on Constraint Handling Rules

Communicated by T. Schrijvers, T. Fruehwirth, F. Raiser

URL: http://www.uni-ulm.de/in/pm/research/events/chr2008

Location: Hagenberg, Austria

The Constraint Handling Rules (CHR) language has become a major declarative specification and implementation language for constraint reasoning algorithms and applications. Algorithms are often specified using inference rules, rewrite rules, sequents, proof rules or logical axioms that can be directly written in CHR. Its clean semantics facilitates program design, analysis and transformation. See the CHR website (http://www.cs.kuleuven.be/~dtai/projects/CHR/) for more information.

Previous Workshops on Constraint Handling Rules were organized in May 2004 in Ulm (Germany), in October 2005 in Sitges (Spain) at ICLP, in July 2006 in Venice (Italy) at ICALP, and in September 2007 in Porto (Portgual) at ICLP.

Topics of Interest
The workshop calls for full papers and short papers describing ongoing work, on all aspects of CHR, including topics such as:

- (Logical) Algorithms
- Applications
- Comparisons with Related Approaches
- Constraint Solvers
- Critical Assessment
- Expressivity and Complexity
- Implementations and Optimization
- Language Extensions (Types, Modules)
- Program Analysis
- Program Transformation and Generation
- Programming Environments (Debugging)
- Programming Pearls
- Retractable Constraints
- Semantics
- Programming Tools
- Language Extensions (Debugging)

Submission Information
All papers must be written in English and not exceed 15 pages in Springer LNCS

format. The authors are encouraged, although not obliged, to submit their papers already in Springer LNCS format. General information about the Springer LNCS series and the LNCS authors' instructions are available at the Springer LNCS/LNAI home page.
Submissions should be sent to chr2008@uni-ulm.de and mention 'CHR 2008 Submission' in the subject. Every submission should include the names and e-mail addresses of the authors (with the corresponding author indicated), the paper abstract in ASCII format and the actual paper in postscript or PDF format. The submission should also indicate whether it is a full paper or a short paper.
Accepted papers will be published in a technical report.

Important dates

- submission: May 5, 2008
- notification of acceptance: June 2, 2008
- final version due: June 16, 2008
- workshop date: July 14, 2008

Organization
Program Committee:

- Francois Fages, INRIA Rocquencourt
- Peter J. Stuckey, NICTA Victoria Laboratory
- Jacques Robin, Universidade Federal de Pernambuco
- Martin Sulzmann, National University of Singapore
- Maurizio Gabbrielli, Universita di Bologna
- Slim Abdennadher, German University in Cairo
- Thom Fruehwirth, UniversitÃ¤t Ulm
- Tom Schrijvers, Katholieke Universiteit Leuven
- Armin Wolf, Fraunhofer FIRST, Berlin
- Veronica Dahl, Simon Fraser University in Vancouver
- Beata Sarna-Starosta, Michigan State University
- Evelina Lamma, UniversitÃ di Ferrara

Workshop Coordinators:
chr2008@uni-ulm.de
Tom Schrijvers (contact person) Department of Computer Science K.U.Leuven
Thom Fruehwirth Faculty of Engineering and Computer Science University Ulm
Frank Raiser Faculty of Engineering and Computer Science University Ulm

Software Announcement
Abella: Interactive theorem proving with lambda-tree syntax

Communicated by Andrew Gacek

URL: http://abella.cs.umn.edu/

I am happy to announce the public release of Abella, an interactive theorem prover that is designed to reason about structural operational semantics style specifications of dynamic and static properties of an object language. Amongst other things, Abella has been used to prove normalizability properties of the lambda calculus, cut-admissibility for a sequent calculus and type uniqueness and subject reduction properties. The most recent successes include solutions to parts 1a and 2a of the POPLmark challenge and a proof of normalizability for the simply-typed lambda-calculus using a logical relations argument in the style of Tait.

Abella is a realization of a two-level logic approach to reasoning in its application domain. One level is defined by a specification logic that supports a transparent encoding of structural operational semantics rules. This logic is a subset of the language of Lambda Prolog and can therefore be animated. The second level, that is called the reasoning logic, embeds the specification logic via definitions of atomic judgments; complicated properties involving these atomic judgments can then be stated and proved in the reasoning logic. An important characteristic of Abella is that it supports the use of lambda-tree syntax in both the specification and the reasoning logics in providing treatments of binding constructs in object language syntax. Reasoning over lambda-tree syntax is supported by the nabla quantifier introduced by Miller and Tiu and the notion of generic judgments. Abella also incorporates a newly developed extension to the notion of definitions of McDowell and Miller that uses the nabla quantifier to encode stronger properties about atomic judgments that are often essential in reasoning tasks.

For more information, the Abella website includes walkthroughs, examples, downloads, and related publications:

http://abella.cs.umn.edu/

The distribution material also contains proofs of the various example properties mentioned in this message.
I welcome your feedback and any questions you may have. Please contact me directly at **andrew.gacek@gmail.com**.

Thank you,
Andrew Gacek

PhD Positions Announcement
Ph.D. Student Positions at Kent University

Communicated by Andy King

Funding is available for the following five PhD studentships within the TCS group at the

University of Kent. Applicants should contact the project supervisor directly for further details.

Project Supervisor: Dr Eerke Boiten (**E.A.Boiten@kent.ac.uk**)
Project Title: Reasoning about Scratch Cards
Scratch cards are used widely in lotteries and games, and recently also in e-voting protocols. However, public confidence in e-voting is very low. This research project can make a difference by developing mathematical and logical abstractions of scratch cards that allow formal reasoning, and consequently watertight proofs of the security of protocols using them. This would be a great project if you are interested in practical symbolic reasoning; knowledge of security, cryptography, formal methods, probability, or logics would be a bonus.

Project Supervisor: Dr Olaf Chitil (**O.Chitil@kent.ac.uk**)
Project Title: Tracing Functional Programs with Hat
Hat (**www.haskell.org/hat**) is a sophisticated tool for locating faults in Haskell programs. Hat consists of a trace generation system plus various tools for viewing a trace. The aim of the research project is to improve Hat by both extending it and easing its application in practise: (1) Apply several theoretical results of a recent EPSRC project on tracing in Hat (e.g. algorithmic debugging with functions as finite maps). (2) Integrate the trace generator of Hat into the byte code interpreter
of the Glasgow Haskell system (GHC). (3) Enable traced code to call and be called from unmodified non-tracing code, such that Hat can use pre-compiled libraries of GHC.

Project Title: The Essence of Transfinite Reductions
Project Supervisor: Dr Stefan Kahrs (**S.M.Kahrs@kent.ac.uk**)
Infinitary Rewriting is an area of Term Rewriting in which research has studied infinitary terms and infinitary reductions. While the notion of infinitary terms is fairly settled, the existing notions of infinitary reduction leave a lot to be desired - the definitions are suspiciously complicated, the established results less than impressive. Thus, there appears to be a lot of room for improvement. There are different angles that are worth exploring. Firstly, there are several alternative ways to define transfinite reductions. Secondly, one would hope that some of these alternative ways lead to good properties of transfinite reduction. Thirdly, it is not even a priori clear what would constitute such a good property.

Project Title: Finding Security Bugs in x86 code
Project Supervisor: Dr Andy King (**A.M.King@kent.ac.uk**)
The project will investigate how security vulnerabilities can be automatically located in x86 code. Rather than trap a fault when it occurs as the program is running, the project will devise compile-time techniques for locating faults before the program is executed. The project will apply techniques from compiling, constraint solving and semantics, though the applicant need not have expertise in all these fields.

Project Title: Refactoring Proofs
Project Supervisor: Prof Simon Thompson (**S.J.Thompson@kent.ac.uk**)
Refactoring allows the programmer to modify the design or structure of a program without changing its behaviour. Recent work in the Functional Programming group at Kent has developed refactoring systems for Haskell 98 (HaRe) and Erlang (Wrangler). Programming and proof have much in common, and indeed under the "propositions as types" analogy, they are different views of the same objects. The aim of this project is

to explore how refactoring can be incorporated into proof development systems, and will combine theoretical work, implementation and usability analysis to ensure that the results will be of value to users of proof assistants. The aim of this project is to investigate refactoring for proofs.

Book Announcement
Constraint Programming Letters - Second Volume

Communicated by Marc van Dongen

URL: http://www.constraint-programming-letters.org/

The Constraint Programming Letters (CSPL) Journal is delighted to announce its second volume, which is dedicated to recent advances in arc consistency.
The volume may be downloaded from the CPL website, which may be found at **http://www.constraint-programming-letters.org/**

Content

- Christophe Lecoutre and Julien Vion. Enforcing Arc Consistency using Bitwise Operations. CPL, 2:21–35, 2008.
- Christophe Lecoutre, Chavalit Likitvivatanavong, Scott G. Shannon, Roland H. C. Yap, and Yuanlin Zhang. Maintaining Arc Consistency with Multiple Residues. CPL, 2: 3–19, 2008.
- Deepak Mehta. Reducing Checks and Revisions in the Coarse-grained Arc Consistency Algorithms. CPL, 2:37–53, 2008.
- M.R.C. van Dongen, A.B. Dieker, and A. Sapozhnikov. The Expected Value and the Variance of the Checks Required by Revision Algorithms. CPL, 2:55–77, 2008.

Program Announcement
European Masters Program in Computational Logic

Communicated by Enrico Franconi

URL: http://www.computational-logic.eu/

The Faculty of Computer Science at the Free University of Bozen-Bolzano (FUB), in Italy (at the heart of the Dolomites mountains in South-Tyrol), is offering the European Masters Program in Computational Logic as part of its Master of

Science in Computer Science offer (Laurea Specialistica). The European Masters Program in Computational Logic is an international distributed Master of Science course, in cooperation with the computer science departments in the following universities:

- Free University of Bozen-Bolzano, Italy
- Technische Universitaet Dresden, Germany
- Universidade Nova de Lisboa, Portugal
- Technische Universitaet Wien, Austria
- Universidad Politecnica de Madrid, Spain

This program, completely in English, involves studying one year at the Free University of Bozen-Bolzano, and completing the second year with a stay in one of the partner universities. After this, the student will obtain, together with the European degree, two Master of Science degrees: the Laurea Specialistica degree from the Free University of Bozen-Bolzano, with legal value in Italy, and the respective Master of Science degree from the visited university, with legal value in its country.

APPLICATION DEADLINES:

- *** 31 May 2008 *** deadline for European and non-European students (notification of acceptance: 15 June 2008)
- 22 August 2008: last deadline only for European students starting at the Free University of Bozen-Bolzano, Italy (notification of acceptance: 5 September 2008)

SCHOLARSHIPS & MONEY SUPPORT:
European citizens can apply to scholarships which are granted purely on the basis of the yearly income of the applicant and of her/his parents or husband/wife. Scholarships may amount up to more than 6,000 EUR per academic year, plus support on the accommodation and total reimbursement of the enrolment fees. These scholarships are also available to non-European citizens with residence in Italy. European students will also get a LLP Socrates Erasmus scholarship for the second year of study abroad, which is 330 EUR per month.

NEW! Every year 10 students with European citizenship can visit Australia (Canberra, Sidney, Melbourne or Brisbane) up to 3 months to work on a research project, sponsored by the European Master. The study period in Australia is part of the study programme and it is fully recognised by the European Master's Program in Computational Logic. The guaranteed scholarship is of 3,100 EUR and it covers the travel and living expenses in Australia.

The KRDB Research Centre offers the annual "IBM & KRDB" awards for the best thesis on a Computational Logic related topic, which is generously sponsored by the IBM Center for Advanced Studies; each winner will receive 500 EUR from IBM. In addition to that, the Italian site in Rome of the IBM Center for Advanced Studies supports scholarships of up to 2,400 EUR to work on a research project or on the thesis at their labs in Rome.

Check the web page for detailed info on other available scholarships:**http://www.computational-logic.eu**

THE STUDY PROGRAMME:

The European Masters Program in Computational Logic is designed to meet the demands of industry and research in this rapidly growing area. Based on a solid foundation in mathematical logic, theoretical computer science, artificial intelligence and declarative programming students will acquire in-depth knowledge necessary to specify, implement and run complex systems as well as to prove properties of these systems. In particular, the focus of instruction will be in deduction systems, knowledge representation and reasoning, artificial intelligence, formal specification and verification, syntax directed semantics, logic and automata theory, logic and computability. This basic knowledge is then applied to areas like logic and natural language processing, logic and the semantic web, bioinformatics, information systems and database technology, software and hardware verification. Students will acquire practical experience and will become familiar in the use of tools within these applications. In addition, students will be prepared for a future PhD, they will come in contact with the international research community and will be integrated into ongoing research projects. They will develop competence in foreign languages and international relationships, thereby improving their social skills.

Applicants should have a Bachelor degree (Laurea triennale) in Computer Science, Computer Engineering, or other relevant disciplines; special cases will be considered. The programme is part of the Master in Computer Science (Laurea Specialistica in Informatica) and it has various strengths that make it unique amongst Italian and European universities:

- Curriculum taught entirely in English: The programme is open to the world and prepares the students to move on the international scene.
- Possibility of a strongly research-oriented curriculum.
- Possibility for project-based routes to obtain the degree and extensive lab facilities.
- Other specialisations with streams in the hottest Computer Science areas, such as Web Technologies, Information and Knowledge Management, Databases and Software Engineering.
- International student community.
- Direct interaction with the local and international industry and research centres, with the possibility of practical and research internships that can lead to future employment.
- Excellent scholarship opportunities and student accommodations.

The European Masters Program in Computational Logic is sponsored scientifically by the European Network of Excellence on Computational Logic (CoLogNET), the European Association of Logic, Language and Information (FoLLI), the European Coordinating Committee for Artificial Intelligence (ECCAI), the Italian Association for Artificial Intelligence (AI*IA), the Italian Association for Informatics (AICA, member of the Council of European Professional Informatics Societies), the Italian Association for Logic and its Applications (AILA), and the Portuguese Association for Artificial Intelligence (APPIA).

THE FREE UNIVERSITY OF BOZEN-BOLZANO:
The Free University of Bozen-Bolzano, founded in 1997, boasts modern premises in the centre of Bozen-Bolzano. The environment is multilingual, South Tyrol being a region where three languages are spoken: German, Italian and Ladin. Studying in a multilingual area has shown that our students acquire the cutting edge needed in the international business world. Many of our teaching staff hails from abroad.

Normal lectures are complemented with seminars, work placements and laboratory work, which give our students a vocational as well as theoretical training, preparing them for their subsequent professional careers. Studying at the Free University of Bozen-Bolzano means, first and foremost, being guided all the way through the student's educational career. Bozen-Bolzano, due to its enviable geographical position in the centre of the Dolomites, also offers our students a multitude of opportunities for spending their free-time. The city unites the traditional with the modern. Young people and fashionable shops throng the city centre where ancient mercantile buildings are an attractive backdrop to a city that is in continual growth. To the south there is the industrial and manufacturing area with prosperous small and medium-sized businesses active in every economic sector. Back in the 17th century Bozen-Bolzano was already a flourishing mercantile city that, thanks to its particular geographic position, functioned as a kind of bridge between northern and southern Europe. As a multilingual town and a cultural centre Bozen-Bolzano still has a lot to offer today. Its plethora of theatres, concerts with special programmes, cinemas and museums, combined with a series of trendy night spots that create local colour make Bozen-Bolzano a city that is beginning to cater for its increasingly demanding student population. And if you fancy a very special experience, go and visit the city's favourite and most famous resident - "Oetzi", the Ice Man of Similaun, housed in his very own refrigerated room in the recently opened archaeological museum.
Bozen-Bolzano and its surroundings are an El Dorado for sports lovers: jogging on the grass alongside the River Talfer-Talvera, walks to Jenesien-S.Genesio and on the nearby Schlern-Sciliar plateau, excursions and mountain climbing in the Dolomites, swimming in the numerous nearby lakes and, last but not least, skiing and snowboarding in the surrounding ski areas.

FURTHER INFORMATION:
Prof. Enrico Franconi or Dr. Sergio Tessaris at **info@fub.computational-logic.eu**

European Masters Program in Computational Logic
Faculty of Computer Science
Free University of Bozen-Bolzano
Piazza Domenicani, 3
I-39100 Bozen-Bolzano BZ, Italy

Phone: +39 0471 016 000
Fax: +39 0471 016 009
Email: **info@fub.computational-logic.eu**
Web site: **http://www.computational-logic.eu**

Summer School Announcement
10th European Agent Systems Summer School

Communicated by Joao Leite

URL: http://centria.di.fct.unl.pt/events/easss08/
LOCATION: New University of Lisbon, Portugal

DATE: 5-9 May, 2008

As its very successful predecessors, EASSS'08 aims to offer a valuable forum for knowledge exchange between various research groups in this field for the benefit of students and researchers at both beginner and advanced level. EASSS consists of a mixture of introductory and advanced courses delivered by internationally leading experts in multi-agent systems, and it covers the full range of theoretical and practical aspects of multi-agent systems.

EASSS'08 will comprise the following courses:

- Introduction to Multiagent Systems
- Logics for Multiagent Systems
- Trust and Reputation in Multiagent Systems
- Service Oriented Agents
- Introduction to Game Theory and Mechanism Design
- Foundations of Institutions
- Agents and Arguments
- Computational Complexity in Multiagent Systems
- Planning in Multiagent Systems
- What Coalitions Can Achieve
- Agent Oriented Software Engineering
- Automated Negotiations in Electronic Markets
- Agent Based Simulation for Social Studies
- Normative Multiagent Systems
- Wireless Sensor Networks and Multiagent Systems
- Agent Swarms Generating Short-term Forecasts and Increasing Situational Awareness

This summer school is open to anyone from academia or industry.
More information can be found at the webpage. Information regarding registration and accomodation will be posted shortly.

Inquiries can be sent to **easss08@gmail.com**

Position Announcement
Research Position in Bioinformatics

Communicated by Pedro Barahona

URL: http://www.eracareers.pt/opportunities/index.aspx?task=global&jobId=8430

Applications are invited for SENIOR RESEARCHER positions in Bioinformatics, in a multidisciplinary project involving Informatics, Structural Biochemistry, Molecular and Cell Biology, Materials Science and Physics. The positions offered are for a 5 year contract to work at CENTRIA, the Centre for Artificial Intelligence in the Faculty of Science of the New University of Lisbon

The candidates must have earned a Ph.D. for more than 3 years, in the areas of machine learning and data mining, constraint programming and optimisation, simulation (including artificial life) or other Artificial Intelligence areas. In exceptional cases, duly justified, we may consider accepting applicants with less than 3 years of post-doctoral experience.

The candidates should be familiar with application of these techniques to bioinformatics applications, namely for sequence matching and comparison, determination of protein structure and interaction, analysis of metabolic pathways, assessment of philogenetic trees, and be able to interact with scientists from within and outside of the Institution, not only from the above mentioned research areas, but also from the areas of Structural Biochemistry and Molecular and Cell Biology.

In addition to such interaction, the candidates are expected to help with post-graduate teaching, including the supervision of post-graduate students, and to write research papers as well as project and grant applications. For any further information contact Prof. Pedro Barahona (pb@di.fct.unl.pt or address below).
Candidates will profit from the technical facilities and expertise available at CENTRIA, at the Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, where they will be stationed.

Please send detailed curriculum vitae and two reference letters to:
Prof. Luís Moniz Pereira
Director
Centro de Inteligência Artificial
Universidade Nova de Lisboa
Quinta da Torre
2825-516 Caparica
Portugal

Ph.D./PostDoc Announcement
Leipzig University

Communicated by Gerhard Brewka

Leipzig University has an open position in a DFG funded project entitled "Defaults and Preferences in Action Formalisms". The position can be filled by a PhD student or a postdoc. Funding is available for 2 years, but it is planned to apply for an extension after the first period. This is a joint project with TU Dresden. Candidates must have a good background in logic based AI.

For further information please contact Gerhard Brewka (brewka AT informatik.uni-leipzig.de).

Prize Announcement
E.W.Beth Dissertation Prize

Communicated by Carlos Areces

URL: http://www.folli.org/

Since 2002, FoLLI (the European Association for Logic, Language, and Information, **www.folli.org**) awards the E. W. Beth Dissertation Prize to outstanding dissertations in the fields of Logic, Language, and Information. We invite submissions for the best dissertation which resulted in a Ph.D. degree in the year 2007. The dissertations will be judged on technical depth and strength, originality, and impact made in at least two of the three fields of Logic, Language, and Computation. Inter-disciplinarity is an important feature of the theses competing for the E. W. Beth Dissertation Prize.

Who qualifies
Nominations of candidates are admitted who were awarded a Ph.D. degree in the areas of Logic, Language, or Information between January 1st, 2007 and December 31st, 2007. There is no restriction on the nationality of the candidate or the university where the Ph.D. was granted. After a careful consideration, FoLLI has decided to accept only dissertations written in English. Dissertations produced in 2007 but not written in English or not translated will be allowed for submission, after translation, also with the call next year (for 2008). Respectively, nominations of full English translations of theses originally written in other language than English and defended in 2006 and 2007 will be accepted for consideration this year, too.

Prize
The prize consists of:

- a certificate
- a donation of 2500 euros provided by the E. W. Beth Foundation.
- an invitation to submit the thesis (or a revised version of it) to the new series of books in Logic, Language and Information to be published by Springer-Verlag as part of LNCS or LNCS/LNAI. (Further information on this series is available on the FoLLI site)

How to submit
Only electronic submissions are accepted. The following documents are required:

1. the thesis in pdf or ps format (doc/rtf not accepted);
2. a ten page abstract of the dissertation in ascii or pdf format;
3. a letter of nomination from the thesis supervisor. Self-nominations are not admitted: each nomination must be sponsored by the thesis supervisor. The letter of nomination should concisely describe the scope and significance of the dissertation and state when the degree was officially awarded;
4. two additional letters of support, including at least one letter from a referee not affiliated with the academic institution that awarded the Ph.D. degree.

All documents must be submitted electronically to **bethaward2008@gmail.com**. Hard copy submissions are not admitted.

In case of any problems with the email submission or a lack of notification within three working days after submission, nominators should write to **goranko@maths.wits.ac.za** or **policriti@dimi.uniud.it**.

Important dates
Deadline for Submissions: April 30th, 2008.
Notification of Decision: July 15th, 2008.

Committee :
Anne AbeillÃ© (UniversitÃ© Paris 7)
Natasha Alechina (University of Nottingham)
Didier Caucal (IGM-CNRS)
Nissim Francez (The Technion, Haifa)
Valentin Goranko (chair) (University of the Witwatersrand, Johannesburg)
Alexander Koller (University of Edinburgh)
Alessandro Lenci (University of Pisa)
Gerald Penn (University of Toronto)
Alberto Policriti (UniversitÃ di Udine)
Rob van der Sandt (University of Nijmegen)
Colin Stirling (University of Edinburgh)

Competition Announcement
The SUMO $100 Challenge

Communicated by Geoff Sutcliffe

The Suggested Upper Merged Ontology (SUMO), the SUMO midlevel (MILO) ontology, and the SUMO domain ontologies, form the largest formal public ontology in existence today. They are being used for research and applications in search, linguistics and reasoning. SUMO is free and owned by the IEEE. The ontologies that extend SUMO are available under GNU General Public License. Adam Pease of Articulate Software is the Technical Editor of SUMO, and the sponsor of the SUMO challenges.

The goal of the SUMO challenges is to verify the consistency of SUMO and its extensions, or, if inconsistency is found, to provide feedback that is sufficient to produce consistency (in a reasonable way). SUMO and its extensions have been translated into the TPTP language, and are included in TPTP v3.4.0 as axiom files in the commonsense reasoning domain. These axiom files form the input for the

challenges. There are three SUMO challenges:

- Verify the consistency or provide feed to repair the base SUMO ontology. This is the axiom file CSR003+0.ax.
- Verify the consistency or provide feed to repair the combined SUMO and MILO ontologies. These are the axiom files CSR003+0.ax and CSR003+1.ax
- Verify the consistency or provide feed to repair the combined SUMO, MILO, and domain ontologies. These are the axiom files CSR003+0.ax to CSR003+25.ax

The winners of the SUMO challenges will each receive $100 in real US dollars, to be awarded at the CADE or IJCAR following successful completion of a challenge. Who says there's no money in ATP?!

Logic-Programming Related Call for Papers

Contents

International Conference on Logic Programming

Date: December 9-13, 2008
Location: Udine, Italy
Submission Deadline: June 9, 2008
URL: http://iclp08.dimi.uniud.it

Principles and Practice of Declarative Programming

Date: July 15-17, 2008
Location: Valencia, Spain
Submission Deadline: April 10, 2008
URL: http://www.clip.dia.fi.upm.es/Conferences/PPDP08

Workshop on Rule-based Programming

Date: July 18, 2008
Location: Hagenberg Castle, Austria
Submission Deadline: April 14, 2008
URL: http://sewiki.iai.uni-bonn.de/rule08/

Workshop on Knowledge Representation for Agents and Multi-Agent Systems

Date: September 16-19, 2008
Location: Sydney, Australia
Submission Deadline: June 1st, 2008
URL: http://www.cs.uu.nl/events/kramas2008/kramas.html

Workshop on Evaluation of Systems for Higher Order Logic

Date: August 10-11, 2008
Location: Sydney, Australia
Submission Deadline: May 19, 2008
URL: http://www.cs.miami.edu/~geoff/Conferences/ESHOL/

Application of Semantic Technologies

Date: September 9th, 2008
Location: Munich, Germany
Submission Deadline: April 25, 2008
URL: http://km.aifb.uni-karlsruhe.de/ws/ast2008

Computer Science Logic

Date: September 15-20, 2008
Location: Bertinoro, Italy
Submission Deadline: March 28, 2008
URL: http://csl2008.cs.unibo.it/

International Conference on Automated Planning and Scheduling

Date: September 15-18, 2008
Location: Sydney, Australia
Submission Deadline: April 28, 2008
URL: http://icaps08.icaps-conference.org/

Workshop on Security and Rewriting Techniques

Date: June 22, 2008
Location: Pittsburgh, USA
Submission Deadline: March 31, 2008
URL: http://www.dsic.upv.es/workshops/secret08

International Verification Workshop

Date: August 10-11, 2008
Location: Sydney, Australia
Submission Deadline: May 15, 2008
URL: http://www.uni-koblenz.de/~beckert/verify08/

Reduction Strategies in Rewriting and Programming

Date: July 14, 2008
Location: Castle of Hagenberg, Austria
Submission Deadline: April 21, 2008

URL: http://cl-informatik.uibk.ac.at/events/wrs08/

Workshop on Logic and Search

Date: Novermber 6-7, 2008
Location: Leuven, Belgium
Submission Deadline: August 15, 2008
URL: http://www.cs.kuleuven.be/~dtai/LaSh08

Foundations of Coordination Languages and Software Architectures

Date: July 13, 2008
Location: Reykjavik, Iceland
Submission Deadline: April 14, 2008
URL: http://foclasa08.lcc.uma.es/

Workshop on Practical Aspects of Automated Reasoning

Date: August 10, 2008
Location: Sydney, Australia
Submission Deadline: May 27, 2008
URL: http://www.eprover.org/EVENTS/PAAR-2008/paar-2008.html

Multi-Agent Systems & Bioinformatics

Date: September 13, 2008
Location: Cagliari, Italy
Submission Deadline: May 18, 2008
URL: http://iasc2.diee.unica.it/masls2008/

International RuleML Symposium

Date: October 30-31, 2008
Location: Orlando, Florida

Submission Deadline: June 2, 2008
URL: http://2008.ruleml.org/

Principles of Knowledge Representation and Reasoning

Date: September 16-19, 2008
Location: Sydney, Australia
Submission Deadline: April 7, 2008
URL: http://www.cse.unsw.edu.au/~kr2008/

Logic-based Program Synthesis and Transformation

Date: July 17-18, 2008
Location: Valencia, Spain
Submission Deadline: May 7, 2008
URL: http://www.informatik.uni-kiel.de/~mh/lopstr08/

International Semantic Web Conference

Date: October 26-30, 2008
Location: Karlsruhe, Germany
Submission Deadline: May 9, 2008
URL: http://iswc2008.semanticweb.org/

Knowledge Representation Ontology Workshop

Date: September 16-19, 2008
Location: Sydney, Australia
Submission Deadline: June 1, 2008
URL: http://www.cse.unsw.edu.au/~kr2008/krow.html

Multidisciplinary Workshop on Advances in Preference Handling

Date: July 13-14, 2008
Location: Chicago, Illinois

Submission Deadline: April 7, 2008
URL: http://wikix.ilog.fr/wiki/bin/view/PreferenceWS/MdPref08

Workshop on Search in Artificial Intelligence and Robotics

Date: July 13-14, 2008
Location: Chicago, Illinois
Submission Deadline: April 7, 2008
URL:
http://www.uwosh.edu/faculty_staff/furcyd/search_symposium_2008

International Workshop on Functional and (Constraint) Logic Programming

Date: July 3-4, 2008
Location: Siena, Italy
Submission Deadline: April 20, 2008
URL: http://wflp08.dimi.uniud.it/

Asian Semantic Web Conference

Date: December 8-11, 2008
Location: Pathumthani, Thailand
Submission Deadline: July 15, 2008
URL: http://www.aswc2008.org/

Joint European Conference on Logics in AI

Date: September 28-October 1, 2008
Location: Dresden, Germany
Submission Deadline: June 2, 2008
URL: http://www.jelia.eu/

Non Monotonic Reasoning Workshop

Date: September 13-16, 2008

Location: Sydney, Australia
Submission Deadline: June 15, 2008
URL: http://www.cse.unsw.edu.au/~kr2008/NMR2008/

Logical Semantic Frameworks with Applications

Date: August 26, 2008
Location: Salvador, Brasil
Submission Deadline: May 18, 2008
URL: http://www.mat.ufmg.br/lsfa2008

Papers to appear in TPLP and TOCL

Contents

Theory and Practice of Logic Programming

http://www.logicprogramming.org/TPLP

Volume 8, Issue 1, January 2008

Regular Papers

- Calculating modules in contextual logic program refinement, Robert Colvin, Ian J. Hayes and Paul Strooper
- Improving Precision of Type Analysis Using Non-Discriminative Union , Lunjin Lu.
- Linear Tabling Strategies and Optimizations Neng-Fa Zhou, Taisuke Sato, and Yi-Dong Shen

Technical Notes

- Recurrence with affine level mappings is P-time decidable for binary CLP (R) Fred Mesnard and Alexander Serebrenik

Programming Pearls:

- Logic programming with satisfiability Michael Codish, Vitaly Lagoon and Peter J. Stuckey

Volume 8, Issue 2, March 2008

Regular Papers

- Experimenting with recursive queries in database and logic programming systems, G. TERRACINA, N. LEONE, V. LIO and C. PANETTA
- Logic programs with monotone abstract constraint atoms, VICTOR W. MAREK, ILKKA NIEMELÄ and MIROSŁAW TRUSZCZYŃSKI

Technical Notes

- Improving Prolog programs: Refactoring for Prolog, ALEXANDER SEREBRENIK, TOM SCHRIJVERS and BART DEMOEN
- A common view on strong, uniform, and other notions of equivalence in answer-set programming, STEFAN WOLTRAN

Book Reviews:

- Learn Prolog Now! Patrick Blackburn, Johan Bos, Kristina Striegnitz College Publications, 2006,
  Bart Demoen
- Constraint Logic Programming using ECLiPSe Krzysztof Apt and Mark Wallace Cambridge University Press, 2007
  Peter J. Stuckey

Accepted Regular Papers

- SWI-Prolog and the web
  JAN WIELEMAKER, ZHISHENG HUANG and LOURENS VAN DER MEIJ
- TCHR: a framework for tabled CLP
  TOM SCHRIJVERS, BART DEMOEN and DAVID S. WARREN
- Translating OWL and semantic web rules into prolog: Moving toward description logic programs
  KEN SAMUEL, LEO OBRST, SUZETTE STOUTENBERG, KAREN FOX, PAUL FRANKLIN, ADRIAN JOHNSON, KEN LASKEY, DEBORAH NICHOLS, STEVE LOPEZ and JASON PETERSON
- Querying XML documents in logic programming
  J. M. ALMENDROS-JIMÉNEZ, A. BECERRA-TERÓN and F. J. ENCISO-BAÑOS
- Query evaluation and optimization in the semantic web
  EDNA RUCKHAUS, EDUARDO RUIZ and MARÍA-ESTHER VIDAL
- Building Rules on Top of Ontologies for the Semantic Web with Inductive Logic Programming
  FRANCESCA A. LISI
- Guarded hybrid knowledge bases
  STIJN HEYMANS, JOS DE BRUIJN, LIVIA PREDOIU, CRISTINA FEIER and DAVY VAN NIEWENBORGH

ACM Transactions on Computational Logic

http://www.acm.org/tocl

The files below are the final versions of the papers submitted by the authors. The definite, published versions of the papers are available from **the TOCL home page** within the ACM Digital Library.

Volume 9, Number 3 (tentative)

- **A Uniform Approach to Constraint-solving for Lists, Multisets, Compact Lists, and Sets** Agostino Dovier, Carla Piazza, and Gianfranco Rossi
- **Foundational Certified Code in the Twelf Metalogical Framework** Karl Crary and Susmit Sarkar
- **Inferring Non-Suspension Conditions for Logic Programs with Dynamic Scheduling** Samir Genaim and Andy King
- **Program Termination and Well Partial Orderings** Andreas Blass and Yuri Gurevich
- **Abstract State Machines Capture Parallel Algorithms: Correction and Extension** Andreas Blass and Yuri Gurevich
- **What Causes a System to Satisfy a Specification?** Hana Chockler, Joseph Y. Halpern and Orna Kupferman
- **Proof Search in Hajek's Basic Logic** Simone Bova and Franco Montagna
- **Conjunctive Query Containment and Answering under Description Logics Constraints** Diego Calvanese, Giuseppe De Giacomo and Maurizio Lenzerini
- **Contextual Modal Type Theory** Aleksandar Nanevski, Frank Pfenning and Brigitte Pientka

Volume 9, Number 4 (tentative)

- **Complexity Results for Security Protocols with Diffie-Hellman Exponentiation and Commuting Public Key Encryption** Yannick Chevalier, Ralf Kuesters, Michael Rusinowitch and Mathieu Turuani
- **Undecidability of the Unification and Admissibility Problems for Modal and Description Logics** Frank Wolter and Michael Zakharyaschev
- **Open Answer Set Programming with Guarded Programs** Stijn Heymans, Davy Van Nieuwenborgh and Dirk Vermeir
- **Reasoning with Recursive Loops under the PLP Framework** Yi-Dong Shen
- **Flat and One-Variable Clauses: Complexity of Verifying Cryptographic Protocols with Single Blind Copying** Helmut Seidl and Kumar Neeraj Verma
- **Verifiable Agent Interaction in Abductive Logic Programming: the SCIFF Framework** Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello and Paolo Torroni

Future Issues (the order of the papers can change)

- **Specifying Norm-Governed Computational Societies** Alexander Artikis, Marek Sergot and Jeremy Pitt
- **Arithmetic Complexity** Lou van den Dries and Yiannis N. Moschovakis
- **Certainty Closure: Reliable Constraint Reasoning with Incomplete or Erroneous Data** Neil Yorke-Smith and Carmen Gervet
- **New Results on Rewrite-based Satisfiability Procedures** Alessandro Armando, Maria Paola Bonacina, Silvio Ranise and Stephan Schulz
- **Reasoning about Actions with Sensing under Qualitative and Probabilistic Uncertainty** Luca Iocchi, Thomas Lukasiewicz, Daniele Nardi and Riccardo Rosati

- **A Finite Equational Base for CCS with Left Merge and Communication Merge** Luca Aceto, Wan Fokkink, Anna Ingolfsdottir and Bas Luttik
- **Logical Characterization of the Counting Hierarchy** Juha Kontinen
- **The Geometry of Linear Higher-Order Recursion** Ugo Dal Lago
- **Probabilistic Bisimulation as a Congruence** Ruggero Lanotte and Simone Tini
- **Extending the LOOP language with Higher-Order Procedural Variables** T. Crolard, E. Polonowski and P. Valarcher
- **Termination of Rewriting Strategies: a Generic Approach** Isabelle Gnaedig and Hélène Kirchner (**Electronic Appendix**)
- **A Compositional Semantics for CHR** Maurizio Gabrielli and Maria Chiara Meo
- **Proofs, Tests and Continuation Passing Style** Stefano Guerrini and Andrea Masini
- **A Flow Calculus of mwp-Bounds for Complexity Analysis** Neil D. Jones and Lars Kristiansen
- **PSPACE Bounds for Rank-1 Modal Logics** Lutz Schröder and Dirk Pattinson
- **Context Semantics, Linear Logic and Computational Complexity** Ugo Dal Lago
- **On the Proof Complexity of Deep Inference** Paola Bruscoli and Alessio Guglielmi
- **A New Function Algebra of EXPTIME Functions by Safe Nested Recursion** Toshiyasu Arai and Naohi Eguchi
- **Checking Timed Büchi Automata Emptiness on Simulation Graphs** Stavros Tripakis

Accepted Papers in Logic-Programming Related Conference

Contents

Theory and Applications of Satisfiability Testing
Guangzhou, China, May 12-15, 2008

http://www.ist.unomaha.edu/padl2008/

Accepted Papers

- Searching for Autarkies to Trim Unsatisfiable Clause Sets
  Mark Liffiton and Karem Sakallah
- A CNF Class Generalizing Exact Linear Formulas
  Stefan Porschen and Ewald Speckenmeyer
- Finding Guaranteed MUSes Fast
  Hans van Maaren and Siert Wieringa
- Complexity and Algorithms for Well-Structured k-SAT Instances
  Konstantinos Georgiou and Periklis A. Papakonstantinou
- Modelling Max-CSP as Partial Max-SAT
  Josep Argelich, Alba Cabiscol, Inês Lynce, and Felip Manyà
- A Max-SAT Inference-Based Pre-processing for Max-Clique
  Federico Heras and Javier Larrosa
- A Preprocessor for Max-SAT Solvers
  Josep Argelich, Chu Min Li, and Felip Manyà
- Towards More Effective Unsatisfiability-Based Maximum Satisfiability Algorithms
  Joao Marques-Silva and Vasco Manquinho
- Designing an Efficient Hardware Implication Accelerator for SAT Solving
  John Davis, Zhangxi Tan, Fang Yu, and Lintao Zhang
- Attacking Bivium Using SAT Solvers
  Tobias Eibach, Enrico Pilz, and Gunnar Völkel
- New Results on the Phase Transition for Random Quantified Boolean

Formulas
Nadia Creignou, Hervé Daudé, Uwe Egly, and Raphaël Rossignol
- Nenofex: Expanding NNF for QBF Solving
  Florian Lonsing and Armin Biere
- Improvements to Hybrid Incremental SAT Algorithms
  Florian Letombe and Joao Marques-Silva
- Random Instances of W[2]-Complete Problems: Thresholds, Complexity, and Algorithms
  Yong Gao
- A Generalized Framework for Conflict Analysis
  G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais
- Adaptive Restart Strategies for Conflict Driven SAT Solvers
  Armin Biere
- Local Restarts
  Vadim Ryvchin and Ofer Strichman
- A Decision-Making Procedure for Resolution-Based SAT-Solvers
  Eugene Goldberg
- Speeding-Up Non-clausal Local Search for Propositional Satisfiability with Clause Learning
  Zbigniew Stachniak and Anton Belov
- SAT Modulo the Theory of Linear Arithmetic: Exact, Inexact and Commercial Solvers
  Germain Faure, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonel
- SAT(ID): Satisfiability of Propositional Logic Extended with Inductive Definitions
  Maarten Mariën, Johan Wittocx, Marc Denecker, and Maurice Bruynooghe
- The OKlibrary: A generative research platform for (generalised) SAT solving
  Oliver Kullmann
- Online Estimation of SAT Solving Runtime
  Shai Haim and Toby Walsh
- How Many Conflicts Does It Need to Be Unsatisfiable?
  Dominik Scheder and Philipp Zumstein
- Computation of Renameable Horn Backdoors
  Stephan Kottler, Michael Kaufmann, and Carsten Sinz
- A New Bound for an NP-Hard Subclass of 3-SAT Using Backdoors
  Stephan Kottler, Michael Kaufmann, and Carsten Sinz
- CNF Encoding for Adjacencies in Boolean Cardinality Constraint
  Sachoun Park and Gihwon Kwon

Coordination Models and Languages
Oslo, Norway, June 4-6, 2008

http://discotec08.ifi.uio.no/pmwiki.php?n=Coordination08.HomePage

Accepted Papers

- Timed Soft Concurrent Constraint Programs
  Francesco Santini, Stefano Bistarelli, Maurizio Gabbrielli and Maria Chiara Meo
- How to infer finite session types in a calculus of services and sessions
  Leonardo Gaetano Mezzina
- Advice for Coordination
  Chris Hankin, Fleming Nielson, Hanne Riis Nielson and Fan Yang
- A formal account of WS-BPEL
  Alessandro Lapadula, Rosario Pugliese and Francesco Tiezzi
- Implementing Session Centered Calculi with IMC
  Lorenzo Bettini, Rocco De Nicola and Michele Loreti
- Formalizing Higher-order Mobile Embedded Business Processes with Binding Bigraphs
  Mikkel Bundgaard, Arne Glenstrup, Thomas Hildebrandt, Espen Højsgaard and Henning Niss
- Actors with Multi-Headed Message Receive Patterns
  Martin Sulzmann, Edmund Lam and Peter Van Weert
- A compositional trace semantics for Orc
  Dimitrios Vardoulakis and Mitchell Wand
- Service Combinators for Farming Virtual Machines
  Karthikeyan Bhargavan, Andy Gordon and Iman Narasamdya
- A coordination model for service-oriented interactions
  João Abreu and José Luiz Fiadeiro
- Encrypted Shared Data Spaces
  Giovanni Russello, Changyu Dong, Naranker Dulay, Michel Chaudron and Maarten Van Steen
- An event-based coordination model for context-aware applications
  Angel Nuñez and Jacques Noyé
- Session Behaviour Types for Orchestration Charts
  Alessandro Fantechi and Elie Najm
- CiAN: A Workflow Engine for MANETs
  Rohan Sen, Catalin Roman and Christopher Gill
- A Process Calculus for Mobile Ad Hoc Networks
  Anu Singh, C. R. Ramakrishnan and Scott A. Smolka
- Multiparty sessions in SOC
  Roberto Bruni, Ivan Lanese, Hernan Melgratti and Emilio Tuosto
- Implementing Joins using Extensible Pattern Matching
  Philipp Haller and Tom Van Cutsem
- Modeling and Analysis of Reo Connectors Using Alloy
  Ramtin Khosravi, Marjan Sirjani, Nesa Asoudeh, Shaghayegh Sahebi and Hamed Iravanchi zadeh
- Alternating-Time Model Checking for Exogenous Coordination
  Sascha Klueppelholz and Christel Baier
- From Flow Logic to Static Type Systems for Coordination Languages
  Rocco De Nicola, Daniele Gorla, Rene Rydhof Hansen, Fleming Nielson, Hanne Riis Nielson, Christian W. Probst and Rosario Pugliese
- Formal analysis of BPMN via a translation into COWS
  Davide Prandi, Paola Quaglia and Nicola Zannone

European Semantic Web Conference
Tenerife, Spain, June 1-5, 2008

http://www.eswc2008.org/

Accepted Papers

- Andre Bolles, Marco Grawunder and Jonas Jacobi. Streaming SPARQL - Extending SPARQL to process data streams
- Glen Hart, Martina Johnson and Catherine Dolbear. Rabbit: Developing a Control Natural Language for Authoring Ontologies
- Yolanda Blanco-Fernandez, José J. Pazos-Arias, Alberto Gil-Solla, Manuel Ramos-Cabrer and Martin Lopez-Nores. Semantic Reasoning: A Path To New Possibilities of Personalization
- Edoardo Pignotti, Peter Edwards, Alun Preece, Nick Gotts and Gary Polhill. Enhancing Workflow with a Semantic Description of Scientist's Intent
- Boontawee Suntisrivaraporn. Module Extraction and Incremental Classification: A Pragmatic Approach for EL+ Ontologies
- Christoph Kiefer, Abraham Bernstein and André Locher. Adding Data Mining Support to SPARQL via Statistical Relational Learning Methods
- Caecilia Zirn, Vivi Nastase and Michael Strube. Distinguishing between Instances and Classes in the Wikipedia Taxonomy
- Jun Zhao, Graham Klyne and David Shotton. Building a Semantic Web Image Repository for Biological Research Images
- Idoia Berges, Jesus Bermudez, Alfredo Goñi and Arantza Illarramendi. Semantic Web technology for Agent Communication Protocols
- Yannis Tzitzikas, Yannis Theoharis and Dimitris Andreou. On Storage Policies for Semantic Web Repositories that Support Versioning
- Eero Hyvönen, Kim Viljanen, Jouni Tuominen and Katri Seppälä. Building a National Semantic Web Ontology and Ontology Service Infrastructure-The FinnONTO Approach
- Zhe Wang, Kewen Wang, Rodney Topor and Jeff Z. Pan. Restricting and forgetting in DL-Lite
- Haofen Wang, Kang Zhang, Qiaoling Liu, Duc Thanh Tran and Yong Yu. Q2Semantic: A Lightweight Keyword Interface to Semantic Search
- Kay-Uwe Schmidt, Jörg Dörflinger, Tirdad Rahmani, Mehdi Sahbi, Susan Thomas and Ljiljana Stojanovic. An User Interface Adaptation Architecture for Rich Internet Applications
- Silvana Castano, Alfio Ferrara, Davide Lorusso, Tobias Henrik Näth and Ralf Moeller. Mapping Validation by Probabilistic Reasoning
- Tomi Kauppinen, Jari Väätäinen and Eero Hyvönen. Creating and Using Geospatial Ontology Time Series in a Semantic Cultural Heritage Portal
- Vassilis Spiliopoulos, Alexandros Valarakos and George Vouros. CSR: Discovering Subsumption Relations for the Alignment of Ontologies
- Ravish Bhagdev, Sam Chapman, Fabio Ciravegna, Vitaveska Lanfranchi and Daniela Petrelli. Hybrid Search: Effectively Combining Keywords and Ontology-based Searches
- Naiwen Lin, Ugur Kuter and Evren Sirin. Web Service Composition with

User Preferences
- Waseem Akhtar, Jacek Kopecky, Thomas Krennwallner and Axel Polleres. XSPARQL: Traveling between the XML and RDF worlds – and avoiding the XSLT Pilgrimage
- Reinhard Pichler, Axel Polleres, Fang Wei and Stefan Woltran. dRDF: Entailment for Domain-restricted RDF
- Heeryon Cho, Toru Ishida, Toshiyuki Takasaki and Satoshi Oyama. Assisting Pictogram Selection with Semantic Interpretation
- Christoph Kiefer and Abraham Bernstein. The Creation and Evaluation of iSPARQL Strategies for Matchmaking
- Matthias Bräuer and Henrik Lochmann. An Ontology for Software Models and its Practical Implications for Semantic Web Reasoning
- Katharina Siorpaes and Martin Hepp. OntoGame: Weaving the Semantic Web by Online Games
- Aurona Gerber, Alta Van der Merwe and Andries Barnard. A Functioal Semantic Architecture
- VinhTuan Thai, Siegfried Handschuh and Stefan Decker. IVEA: An Information Visualization Tool for Personalized Exploratory Document Collection Analysis
- Bastian Quilitz and Ulf Leser. Querying Distributed RDF Data Sources with SPARQL
- Stefan Dietze, Alessio Gugliotta and John Domingue. Conceptual Situation Spaces for Semantic Situation-Driven Processes
- Antoine Isaac, Henk Matthezing, Lourens van der Meij, Stefan Schlobach, Shenghui Wang and Claus Zinn. Putting ontology alignment in context: usage scenarios, deployment and evaluation in a library case
- Valentin Tablan, Danica Damljanovic and Kalina Bontcheva. A Natural Language Query Interface to Structured Information
- Kinga Schumacher, Michael Sintek and Leo Sauermann. Combining Fact and Document Retreival with Spreading Activation for Semantic Desktop Search
- Tudor Groza, Siegfried Handschuh, Knud Möller and Stefan Decker. KonneX-SALT: First Steps towards a Semantic Claim Federation Infrastructure
- Dumitru Roman, Michael Kifer and Dieter Fensel. WSMO Choreography: From Abstract State Machines to Concurrent Transaction Logic
- Andreas Langegger, Wolfram Wöß and Martin Blöchl. A Semantic Web middleware for Virtual Data Integration on the Web
- Paolo Bouquet, Heiko Stoermer and Barbara Bazzanella. An Entity Naming System (ENS) for the Semantic Web
- Gunnar Grimnes, Peter Edwards and Alun Preece. Istance Based clustering of Semantic Web Resources
- Claudia d'Amato, Nicola Fanizzi and Floriana Esposito. Query Answering and Ontology Population: an Inductive Approach
- Anthony Ventresque, Sylvie Cazalens, Philippe Lamarre and Patrick Valduriez. Improving interoperability using query interpretation in semantic vector spaces
- Ernesto Jimenez-Ruiz, Bernardo Cuenca Grau, Ulrike Sattler, Thomas Schneider and Rafael Berlanga-Llavori. Safe and Economic re-use of ontologies: a logic-based methodology and tool support
- Nicola Fanizzi, Claudia d'Amato and Floriana Esposito. Conceptual

Clustering and its Application to Concept Drift and Novelty Detection
- Tomas Vitvar, Jacek Kopecky, Jana Viskova and Dieter Fensel. WSMO-Lite Annotations for Web Services
- Mauricio Espinoza, Asunción Gómez-Pérez and Eduardo Mena. Enriching an Ontology with Multilingual Information
- Riccardo Rosati. Finite model reasoning in DL-Lite
- Carlos Pedrinaci, John Domingue and Ana Karla Alves de Medeiros. A Core Ontology for Business Process Analysis
- Sebastian Dietzold, Jörg Unbehauen and Sören Auer. xOperator - Interconnecting the Semantic Web and Instant Messaging Networks
- Simon Scerri, Siegfried Handschuh and Stefan Decker. Semantic Email as a communication medium for the Social Semantic Desktop
- Angela Maduko, Kemafor Anyanwu, Amit Sheth and Paul Schliekelman. Graph Summaries for Subgraph Frequency Estimation
- Laura Hollink, Mark van Assem, Antoine Isaac, Shenghui Wang and Guus Schreiber. Two Variations on Ontology Alignment Evaluation: Methodological Issues
- Richard Cyganiak, Renaud Delbru, Holger Stenzhorn, Giovanni Tummarello and Stefan Decker. Semantic Sitemaps: Efficient and Flexible Access to Datasets on the Semantic Web
- Dimitrios Kourtesis and Iraklis Paraskakis. Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery

---

Declarative Agent Languages and Technology
Estoril, Portugal, May 12, 2008

http://www.di.unito.it/%7Ebaldoni/DALT-2008/

Accepted Papers

- A complete STIT logic for knowledge and action, and some of its applications
  Jan Broersen
- Combining Multiple Knowledge Representation Technologies into Agent Programming Languages
  Mehdi Dastani, Koen V. Hindriks, Peter Novak, and Nick Tinnemeier
- Model-checking strategic ability and knowledge of the past of communicating coalitions
  Dimitar P. Guelev and Catalin Dima
- JASDL: A Practical Programming Approach Combining Agent and Semantic Web Technologiess
  Thomas Klapiscak and Rafael H. Bordini
- Leveraging new plans in AgentSpeak(PL)
  Felipe Meneguzzi and Michael Luck
- Increasing bid expressiveness for effective and balanced e-barter trading
  Azzurra Ragone, Tommaso Di Noia, Eugenio Di Sciascio, and Francesco

M. Donini
- Inductive Negotiation in Answer Set Programming
  Chiaki Sakama
- Mental State Abduction of BDI-Based Agents
  Michal P. Sindlar, Mehdi M. Dastani, Frank Dignum, and John-Jules Ch. Meyer
- Iterated Belief Revision In the face of Uncertain Communication
  Yoshitaka Suzuki, Satoshi Tojo, and Stijn De Saeger
- Abstracting and Verifying Strategy-proofness for Auction Mechanisms
  Emmanuel M. Tadjouddine, Frank Guerin, and Wamberto Vasconcelos

Short presentations

- Formalising Proactive Maintenance Goals
  Simon Duff and James Harland
- Using Temporal Logic to integrate Goals and Qualitative Preferences into Agent Programming
  Koen V. Hindriks and M. Birna van Riemsdijk
- A Framework for Agent Communication based on Goals and Argumentation
  Mohamed Mbarki, Jamal Bentahar, John-Jules Meyer, and Bernard Moulin
- Agent Communicability in Belief Update Logic
  Mikito Kobayashi and Satoshi Tojo

Temporal Representation and Reasoning
Montreal, Canada, June 16-18, 2008

http://www.time2008.org/

Accepted Papers

- Topology-based Variable Ordering Strategy for Solving Disjunctive Temporal Problems
  Yuechang Liu, Yunfei Jiang
- Efficient Bit-Level Model Reductions for Automated Hardware Verification
  Sergey Tverdyshev, Eyad Alkassar
- The complexity of CARET + Chop
  Laura Bozzelli
- Good friends are hard to find!
  Thomas Brihaye, Mohamed Ghannem, Nicolas Markey, Lionel Rieg
- TLP-GP : solving temporally-expressive planning problems
  Frederic Maris, Pierre Regnier
- Efficient Spatio-temporal Similarity Join of Large Sets of Moving Object Trajectories
  Hui Ding, Goce Trajcevski, Peter Scheuermann
- Practical First-Order Temporal Reasoning

Michael Fisher, Clare Dixon, Alexei Lisitsa, Boris Konev
- A Labeled Tableaux System for the Distributed Temporal Logic DTL
  Luca Vigano, David Basin, Carlos Caleiro, Jaime Ramos
- Labeled Natural Deduction Systems for a Family of Tense Logics
  Luca Vigano, Marco Volpe
- An optimal tableau for Right Propositional Neighborhood Logic over trees
  Pietro Sala, Davide Bresolin, Angelo Montanari
- Decomposition of Decidable First-Order Logics over Integers and Reals
  Florent Bouchy, Jérôme Leroux, Alain Finkel

Accepted short papers

- Regarding Overlapping as a Basic Concept of Subset Spaces
  Bernhard Heinemann
- A Heuristic Approach to Order Events in Narrative Texts
  Farid Nouioua
- Satisfying a Fragment of XQuery by Branching-Time Reduction
  Sylvain Hallé, Roger Villemaire
- Moving Spaces
  Michael Winter, Ivo Duentsch
- Time Aware Mining of Itemsets
  Bashar Saleh, Florent Masseglia
- A Greedy Approach Towards Parsimonious Temporal Aggregation
  Juozas Gordevicius, Johann Gamper, Michael Boehlen
- Towards a Formal Framework for Spatio-Temporal Granularities
  Gabriele Pozzani, Carlo Combi, Alberto Belussi
- Representing Public Transport Schedules as Repeating Trips
  Romans Kasperovics, Michael Boehlen, Johann Gamper

Computability in Europe
Athens, Greece, June 15-20, 2008

http://www.cs.swan.ac.uk/cie08/

Accepted Papers

- Grazyna Zwozniak Induced matchings in graphs of maximum degree three
- Keita Yokoyama Reverse Mathematics for Fourier Expansion
- Guohua Wu and Jiang Liu Joining to High Degrees
- Joost Winter Space Complexity in Ordinal Turing Machines
- Michael Weiss and Gregory Lafitte Simulations Between Tilings
- Daniel Ventura, Mauricio Ayala-Rincon and Fairouz Kamareddine Principal Typings for Explicit Substitutions Calculi
- Alina Vasilieva Quantum Query Algorithms for AND and OR Boolean

Functions
- Guillaume Theyssier and Mathieu Sablik Topological Dynamics of 2D Cellular Automata
- Hayato Takahashi and Kazuyuki Aihara Probabilistic machines vs. relativized computation
- Ákos Tajti and Benedek Nagy Solving Tripartite Matching by Interval-valued Computation in Polynomial Time
- Kohtaro Tadaki A statistical mechanical interpretation of algorithmic information theory
- Mariya Ivanova Soskova Cupping Classes of Sigma 2 Enumeration Degrees
- Alexandra Soskova Omega Degree Spectra
- Michael Soltys and Craig Wilson On the complexity of computing winning strategies for finite poset games
- Boris Solon Almost partial m-reducibility
- Victor Selivanov and Klaus W. Wagner Complexity of aperiodicity for topological properties of regular ω-languages
- Zenon Sadowski Optimal proof systems and complete languages
- Krishna S, Lakshmi Manasa G and Kumar Nagaraj Updatable Timed Automata with Additive and Diagonal Constraints
- Krishna S and Gabriel Ciobanu On the Computational Power of Enhanced Mobile Membranes
- Vladimir Rybakov and Sergei Babyonyshev Decidability of Hybrid Logic with Local Common Knowledge based on Linear Temporal Logic LTL
- Alexandra Revenko Autostability of Automatic Linear Orders
- Daowen Qiu Simulations of Quantum Turing Machines by Quantum Multi-Counter Machines
- Vadim Puzarenko Properties on Admissible Sets
- Mihai Prunescu Polynomial iterations over finite fields
- Petrus H. Potgieter Computable counter-examples to the Brouwer fixed-point theorem
- Sergei Podzorov Upper Semilattices in Many-One Degrees
- Camelia Pintea, Petrica Claudiu Pop, Camelia Chira, Dan Dumitrescu and Corina Pop Sitar An efficient ant colony optimization algorithm for solving the airport gate assignment problem
- Yongri Piao, Seoktae Kim and Sung-Jin Cho Two-Dimensional Cellular Automata Transforms for a Novel Edge Detection
- Florian Pelupessy and Andreas Weiermann Classifying the phase transition threshold for unordered regressive Ramsey numbers
- Nicolas Ollinger Two-by-two Substitution Systems and the Undecidability of the Domino Problem
- Vivek Nigam Using Tables to Construct Non-Redundant Proofs
- Takako Nemoto Complete Determinacy and Subsystems of Second Order Arithmetic
- Marcin Mostowski Limiting recursion, FM--representability, and hypercomputations
- Russell Miller and Dustin Mulcahey Perfect Local Computability and Computable Simulations
- Barnaby Martin First-Order Model Checking Problems Parameterized by the Model
- Angelos Manousaridis, Michalis Papakyriakou and Nikolaos Papaspyrou

From Program Verification to Certified Binaries
- Johann Makowsky From Hilbert's Program to a Logic Toolbox
- Iris Loeb Factoring out Intuitionistic Theorems: Continuity Principles and the Uniform Continuity Theorem
- Chung-Chih Li Query-Optimal Oracle Turing Machines for Type-2 Computations
- Maikel Leon, Isis Bonet and Zenaida Garcia Combining Concept Maps and Petri Nets to Generate Intelligent Tutoring Systems
- Stephane Le Roux Discrete Non Determinism and Nash Equilibria for Strategy-Based Games
- James Lathrop, Jack H. Lutz, Matthew J. Patitz and Scott M. Summers Computability and Complexity in Self-Assembly
- Sunil Kothari and James Caldwell On Extending Wand's Type Reconstruction Algorithm to Handle Polymorphic Let
- Margarita Korovina and Nicolai Vorobjov Safety Properties Verification for Pfaffian Dynamics
- Peter Koepke and Russell Miller An Enhanced Theory of Infinite Time Register Machines
- Bakhadyr Khoussainov, Michael Brough and Peter Nelson Sequential Automatic Algebras
- Basil A. Karadais A Plotkin Definability Theorem for Atomic-Coherent Information Systems
- Temesghen Kahsai and Marino Miculan Implementing Spi Calculus using Nominal techniques
- Reinhard Kahle Towards Reverse Proofs-as-Programs
- Herman Ruge Jervell Ordering finite labeled trees
- ilyes jenhani, Zied Elouedi and Salem Benferhat The Use of Information Affinity in Possibilistic Decision Tree Learning and Evaluation
- Maurice Jansen A Non-Linear Lower Bound for Syntactically Multilinear Algebraic Branching Programs
- Sanjay Jain, Frank Stephan and Nan Ye Prescribed Learning of Indexed Families
- Gabriel Istrate, Madhav Marathe and S. S. Ravi Adversarial Scheduling Analysis of Game-Theoretic Models of Norm Diffusion
- Bernhard Irrgang and Benjamin Seyfferth Multitape Ordinal Machines and Primitive Recursion
- Yoon-Hee Hwang, Sung-Jin Cho, Un-Sook Choi and Han-Doo Kim Modelling Linear Cellular Automata with the minimum stage corresponding to CCSG based on LFSR
- Yoshihiro Horihata and Keita Yokoyama Singularities of Holomorphic Functions in Subsystems of Second Order Arithmetic
- Mircea-Dan Hernest and Paulo Oliva Hybrid Functional Interpretations
- LI Hengwu Approximating Degree-Bounded Minimum Spanning Trees of Directed Acyclic Graphs
- Emmanuel Hainry Reachability in linear dynamical systems
- Xiaoyang Gu and Jack H. Lutz Effective Dimensions and Relative Frequencies
- Rica Gonen On the Hardness of Truthful Online Auctions with Multidimensional Constraints
- Walid Gomaa Expressibility of Divisibility inside Σ_1^1
- Christian Glaßer, Christian Reitwießner and Victor Selivanov The

Shrinking Property for NP and coNP
- Alexander Gavryushkin Computable Models Spectras of Ehrenfeucht Theories
- Christine Gaßner Computation over Groups
- André Luiz Galdino and Mauricio Ayala-Rincon Verification of Newman's and Yokouchi's Lemmas in PVS
- willem fouche Subrecursive complexity of identifying the Ramsey structure of posets
- Ekaterina Fokina Algorithmic Properties of Structures for Languages with Two Unary Functional Symbols
- Pantelis Eleftheriou, Costas Koutras and Christos Nomikos Notions of Bisimulation for Heyting-Valued Modal Languages
- Jérôme Durand-Lose Abstract geometrical computation: beyond the Blum, Shub and Smale model with accumulation
- Jacques Duparc and Alessandro Facchini Describing the Wadge Hierarchy for the Alternation Free Fragment of μ-Calculus (I): The Levels Below ω_{1}
- Giovanni Di Crescenzo and Helger Lipmaa Succinct NP Proofs from an Extractable-Algorithm Assumption
- Michiel De Smet and Andreas Weiermann Phase transitions for weakly increasing sequences, the Erdös-Szekeres theorem and the Dilworth theorem
- Liesbeth De Mol and Maarten Bullynck A week-end off. The first extensive number-theoretical computation on the ENIAC.
- Gregorio de Miguel Casado, Juan Manuel García-Chamizo and Higinio Mora Mora Online-division with Periodic Rational Numbers
- Barbara Csima, Jiamou Liu and Bakhadyr Khoussainov Computable Categoricity of Graphs with Finite Components
- Antonio Carlos Costa and Graçaliz Dimuro Introducing Service Schemes and Systems Organization in the Theory of Interactive Computation
- Charalampos Cornaros Pell equations and weak regularity priciples
- Sung-Jin Cho, Un-Sook Choi, Han-Doo Kim, Yoon-Hee Hwang and Jin-Gyoung Kim Phase shifts of LFSM as pseudorandom number generators for BIST for VLSI
- Luca Chiarabini Extraction of Efficient Programs from Correct Proofs: The Case of Structural Induction over Natural Numbers
- William Calhoun Triviality and Minimality in the Degrees of Monotone Complexity
- Jérémie Cabessa and Jacques Duparc The Algebraic Counterpart of the Wagner Hierarchy
- Amir Ben-Amram, Lars Kristiansen and Neil Jones Linear, Polynomial or Exponential? Complexity Inference in Polynomial Time
- Edwin Beggs and Annelies Gerber Algorithms for Analytic Functions and Applications to Toeplitz Operators
- Mathias Barra Pure iteration and periodicity
- George Metcalfe and Matthias Baaz Herbrand Theorems and Skolemization for Prenex Fuzzy Logics
- Tiago Azevedo, Mario Benevides, Fabio Protti and Marcelo Sihman On detecting deadlock in the Pi-Calculus
- Argimiro Arratia and Iain Stewart Program schemes with deep pushdown storage

- Kostyantyn Archangelsky A new conception of the Euclidian algorithm for the determination of the least common multiple of formal power series
- Luis Antunes and André Souto Sophisticated Infinite Sequences
- Bogdan Aman and Gabriel Ciobanu Decidability Results for Mobile Membranes derived from Mobile Ambients
- Marco Almeida, Nelma Moreira and Rogério Reis On the performance of automata minimization algorithms

---

European Symposium on Programming
Budapest, Hungary, March 29-April 6, 2008

http://esop2008.doc.ic.ac.uk/

Accepted Papers

- A Sound Semantics for OCaml$_{light}$
  Scott Owens
- Parametric polymorphism through run-time sealing, or Theorems for low, low prices!
  Jacob Matthews and Amal Ahmed
- Regular Expression Subtyping for XML Query and Update Languages
  James Cheney
- A Theory of Hygienic Macros
  David Herman and Mitchell Wand
- A Hybrid Denotational Semantics for Hybrid Systems
  Olivier Bouissou and Matthieu Martel
- Full Abstraction for Linda
  Cinzia Di Giusto and Maurizio Gabbrielli
- Practical Programming with Higher-Order Encodings and Dependent Types
  Adam Poswolsky and Carsten Schürmann
- Programming in JoCaml
  Louis Mandel and Luc Maranget
- Playing with Toy: Constraints and Domain Cooperation
  Sonia Estévez, Antonio J. Fernández and Fernando Saenz-Perez
- Typing safe deallocation
  Gerard Boudol
- Iterative Specialisation of Horn Clauses
  Christoffer Rosenkilde Nielsen, Flemming Nielson and Hanne Riis Nielson
- Ranking Abstractions
  Aziem Chawdhary, Byron Cook, Sumit Gulwani, Mooly Sagiv and Hongseok Yang
- Non-disjunctive Numerical Domain for Array Predicate Abstraction
  Xavier Allamigeon
- Upper Adjoints for Fast Inter-procedural Variable Equalities
  Markus Müller-Olm and Helmut Seidl

- Cover Algorithms and their Combination
  Sumit Gulwani and Madanlal Musuvathi
- Trust and Authorization via Provenance and Integrity in Distributed Objects
  Andrew Cirillo, Radha Jagadeesan, Corin Pitcher and James Riely
- Linear Declassification
  Yuta Kaneko and Naoki Kobayashi
- Just Forget It - The Semantics of Enforcement of Information Erasure
  Sebastian Hunt and David Sands
- Open Bisimulation for the Concurrent Constraint Pi-Calculus
  Maria Grazia Buscemi and Ugo Montanari
- The Conversation Calculus: A Model of Service Oriented Computation
  Hugo Vieira, Luis Caires and Joao Seco
- Inferring Channel Buffer Bounds via Linear Programming
  Tachio Terauchi and Adam Megacz
- Verification of Equivalent-Results Methods
  Rustan Leino and Peter Müller
- Semi-Persistent Data Structures
  Sylvain Conchon and Jean-Christophe Filliatre
- A Realizability Model for Impredicative Hoare Type Theory
  Rasmus L. Petersen, Lars Birkedal, Aleksandar Nanevski and Greg Morrisett
- Oracle Semantics for Concurrent Separation Logic
  Aquinas Hobor, Andrew Appel and Francesco Zappa Nardelli
- Certificate~Translation in Abstract Interpretation
  Gilles Barthe and César Kunz
- A formal implementation of value commitment
  Cédric Fournet, Nataliya Guts and Francesco Zappa Nardelli

Net Talk

edited by Roberto Bagnara

Content:

Advice from a newbie
From: **anyway <b@rt.luna.be>**

Dear comp.lang.prolog,

Sorry this is long, but I really have important things to show ...

I recently got my hands on Prolog. I really think I got the hang of it: I can do anything I used to do in my (other) most favourite languages, and often the Prolog version is even more concise, although probably somewhat obscure to the non-informed. And to my surprise (well, actually not really :-), I have come up with versions of common predicates that apparently nobody conceived of before: here is my version of the reverse predicate - I named it rev/2 because it could otherwise clash with something in the library:

```
rev(List,_) :-
        member(X,List),
        asserta(was_in_list(X)),
        fail.
rev(_,RevList) :-
        findall(X,retract(was_in_list
(X)),RevList).
```

I am extremely proud of the combination of using an aggregate predicate and a database predicate in the last clause. But see later for some damper on my hapiness.

I also have a brand new version of append - named app/3 for the same reason:

```
app(L1,L2,_) :-
        assert(result(L2)),
        member(X,L1), asserta(was_in_list(X)),
        fail.
```

```
app(_,_,_) :-
        retract(was_in_list(X)),
        retract(result(PartialRes)),
        assert(result([X|PartialRes])),
        fail.
app(_,_,L3) :-
        retract(result(L3)).
```

I know it uses one more clause than the usual definition, but to be frank, I think my version is much easier to understand: for one thing, it avoids recursion (showing it is really unnecessary) and have you noticed how I cunningly exploit the logical update view of dynamic predicates in the second clause ? It is great that all things fit together so nicely.

I actually like my version of append better than my version of reverse, because I am inclined to the following implementation schema (call it a design pattern if you want):

one (or more) clause puts the data in the database [where it belongs naturally]
one (or more) clause does the actual work - all in the database of course
one clause picks up the result from the database

app/3 does this beautifully - rev/2 probably needs some work to achieve this ideal.

As you can tell, I am very taken in by Prolog, but I do have some quibbles with Prolog as a language, its implementors and its designers.

1) maybe you find my expercience too limited, but my versions of rev and app show clearly that the sequence

```
member(X,L), asserta(foo(X))
```

is very useful: it is worth lifting this to the status of a design pattern and have it abbreviated to something like

```
FORALL X IN L ASSERT foo(X)
```

What do people think about this ? [I am transferring stuff from my other favourite languages - in particular COBOL - to Prolog here, but this will help wider acceptance of Prolog, I am sure]

2) let's focus on my definition of rev for now (the app version has the

same issue): it works fine for lists of atoms, or integers, mixed (atoms and integers) lists and even lists whose elements are any term without variables (some textbooks name these terms ground, like coffee :-); but when I reverse a list with terms with variables, then these variables become disconnected; here is an example:

```
?- rev([X],[Y]), X == Y.
No
```

I have checkd this result in SWI, GNU, XSB, CIAO, ECLIPSE, SICStus and B-Prolog: always the same No. I then checked the ISO Prolog standard and it seems that this is indeed what is required: assert/retract disconnect variables ! While this might help a logical reading of those builtins, I much prefer a version of assert/retract that would keep the connection of variables, so that my rev/2 (and app/3) work as intended

3) let's now focus on my app/3: somehow every Prolog implementation I tried got the complexity of my predicate wrong; my code is cleary O(n) with n the length of the first argument; but I have noticed that Prolog systems make it into O(n*n) in practice; that's no good; no wonder Prolog is hardly used - please Prolog implementors, do a favour to your language and get this right !

4) both my rev/2 and app/3 exhibit an even stranger thing: they are also linear in the size of the ELEMENTS of the involved lists; everyone with half a brain just knows that append and reverse are by nature independent of the size of the list elements - so why are Prolog implementations getting this one wrong as well ? why is it too difficult to get polymorphic types right for you ?

5) the following is really a bummer: I expect a predicate to be callable at all times, even in the middle of some other complicated code; what I mean is this - illustrating it on app/3 and rev/2: I will call rev([1,2,3],RevL) in the middle of a call to app/3 by the following code for append

```
app(L1,L2,_) :-
        assert(result(L2)),
        member(X,L1), asserta(was_in_list(X)),
        fail.
app(_,_,_) :-
        rev([1,2,3],RevL), write(RevL), nl,
        retract(was_in_list(X)),
        retract(result(PartialRes)),
        assert(result([X|PartialRes])),
```

```
        fail.
app(_,_,L3) :-
        retract(result(L3)).
```

Now imagine the following call and answer (in SWI - but again consistent in other Prolog systems):

```
       ?- app([a,b,c],[d,e],L).
       [3, 2, 1, c, b, a]

       L = [d, e]
```

Hard to believe, but there you go ! Apparently, the added call to rev/2 operates in the same dynamic predicate space as the toplevel call to app/3. For me this almost blew it. Luckily the fix is easy: Prolog should encapsulate its asserts/retracts in a dynamicaly scoped way. Shouldn't be too difficult for Prolog implementors to accomodate for such a nice principle, I'd say.

6) The following made me really laugh: none of the predicates I wrote (including rev/2 and app/3) work backwards, i.e. ?- app(X,Y, [1,2]). does not split [1,2] in all possible pairs of lists making up [1,2]. Still, all Prolog textbooks cannot stop talking about this feature.

It seems that all this fuzz about bidirectional programming in Prolog is just a myth. No big deal for me: I will not miss it. But why such bad marketing ?

Before I go, I want to share with you my version of merge/3 (named merg/2):

```
merg(L1,_,_) :- member(X,L1), assertz(list1(X)),
fail.
merg(_,L2,_) :- member(X,L2), assertz(list2(X)),
fail.
merg(_,_,_) :-
        retract(list1(X)),
        assertz(result(X)),
        (retract(list2(Y)) ->
            assertz(result(Y))
        ;
            true
        ),
        fail.
merg(_,_,_) :-
        retract(list2(X)),
```

```
        assertz(result(X)),
        fail.
merg(_,_,L3) :-
        findall(X,retract(result(X)),L3).
```

My design patterns work fantastic (apart from the problems reported above), don't you think ?

I do have some more advice for the Prolog community, in particular about the cut (!/0) whose scope I find rather limited, but maybe more about that later: this was enough for one day's work. I am off now, to refactor some more predicates. On my todo list are the inverse of merg/3, and I would really like a version of member/2 which does not come from the library. I have seen the library version of member/2 and it is extremely recursive: it is both left and right recursive. Probably Prolog programmers don't even know this, or else they would avoid it as the plague: how can you be sure it works correctly ? It must be based on an weird programming paradigm from the seventies or maybe even the sixties. Anyway, if you feel you can contribute, please do so in this newsgroup: together we can improve the world, or at least Prolog !

My blessings,

b@rt.luna.be

From: **Chip Eastham**
Subject: Advice from a newbie

anyway <b@rt.luna.be> wrote:

```
> Dear comp.lang.prolog,
>
> Sorry this is long, but I really have important things to
show ...
>
> I recently got my hands on Prolog.
```

Welcome to the fray!

```
> rev(List,_) :-
>         member(X,List),
>         asserta(was_in_list(X)),
>         fail.
```

```
> rev(_,RevList) :-
>         findall(X,retract(was_in_list(X)),RevList).
>
> I am extremely proud of the combination of using an
aggregate
> predicate and a database predicate in the last clause.
But see later
> for some damper on my hapiness.
```

[snip]

Based on the following comments in your post, you seem to mistrust recursion and feel that Prolog would be better off without it. I don't think I've encountered this point of view before...

As you experimented, you found that asserting dynamic facts does not preserve the semantics of variable identification in the same way as their unification in rule chaining. That's a keen observation on your part, and I look forward to seeing if your opinion of recursion (and polymorphism??) improves with practice.

```
> I do have some more advice for the Prolog community, in
particular
> about the cut (!/0) whose scope I find rather limited,
but maybe more
> about that later: this was enough for one day's work. I
am off now, to
> refactor some more predicates. On my todo list are the
inverse of
> merg/3, and I would really like a version of member/2
which does not
> come from the library. I have seen the library version of
member/2 and
> it is extremely recursive: it is both left and right
> recursive. Probably Prolog programmers don't even know
this, or else
> they would avoid it as the plague: how can you be sure it
works
> correctly ? It must be based on an weird programming
paradigm from
> the seventies or maybe even the sixties.
```

[snip]

There are generalizations to the cut construction in various implementation dependent ways. One keyword to search on is "getBackTrack" (resp. cutBackTrack), which are useful in implementing Prolog within Prolog, among other things.

I'm not sure what you mean by saying the standard version of member/2 is "both left and right recursive." Here:

```
member(H,[H|_]).
member(X,[_|T]) :- member(X,T).
```

This is "tail recursive" in the sense that when the outer instance of the member/2 goal invokes the inner instance, it's the final subgoal and there are no "open" backtrack points to consider. So it can be implemented by the Prolog engine in an efficient manner. What's not to like?
.

From: **anyway <b@rt.luna.be>**
Subject: Advice from a newbie

Chip Eastham wrote:
```
> Based on the following comments in your post, you
> seem to mistrust recursion and feel that Prolog
> would be better off without it.  I don't think
> I've encountered this point of view before...
```

Are you condescending on purpose, or do you have no clue what I mean ?

```
> As you experimented, you found that asserting
> dynamic facts does not preserve the semantics
> of variable identification in the same way as
> their unification in rule chaining.  That's a
> keen observation on your part, and I look
> forward to seeing if your opinion of recursion
> (and polymorphism??) improves with practice.
```

Same question. Or maybe you are just pulling my leg.

```
> There are generalizations to the cut construction
> in various implementation dependent ways.  One keyword
> to search on is "getBackTrack" (resp. cutBackTrack),
> which are useful in implementing Prolog within
> Prolog, among other things.
```

Those are quite unimaginative and known in various implementations, e.g., in XSB they are named '_$savecp'/1 and '_$cutto'/1, and I could imagine that any Prolog implementation must have those, even if they are not in the manual. I was thinking about something more far reaching than that.

```
> I'm not sure what you mean by saying the standard version
> of member/2 is "both left and right recursive."  Here:
>
> member(H,[H|_]).
> member(X,[_|T]) :- member(X,T).
>
```

```
> This is "tail recursive" in the sense that when
> the outer instance of the member/2 goal invokes
> the inner instance, it's the final subgoal and
> there are no "open" backtrack points to consider.
> So it can be implemented by the Prolog engine in
> an efficient manner.  What's not to like?
```

It seems you do not know what I mean by left- and right-recursive. Maybe I should rephrase that to head- and tail-recursive. What is an "open" backtrack point ? Do there exist "closed" backtrack points as well then ? You seem to be an expert in Prolog implementation: what can the Prolog engine (PE ?) do so efficiently for tail-recursive calls it cannot do for others ? I also find the terminology "inner and outer instance of a goal" quite strange: no Prolog textbook I have read mentions it.

Now all the above is just chit-chat: can we focus on the real contribution of my original post ? The database stuff, the design patterns, the language design advice ? Please, I do not want to stall !

From: **Chip Eastham**
Subject: Advice from a newbie

anyway <b@rt.luna.be> wrote:

```
>> Based on the following comments in your post, you
>> seem to mistrust recursion and feel that Prolog
>> would be better off without it.  I don't think
>> I've encountered this point of view before...
>
> Are you condescending on purpose, or do you have no clue
what I mean ?
```

If I'd written what you wrote, this is how I'd want someone to respond to me. I tried to respectfully and accurately restate the main point, as it seems to me, of your post. Possibly I failed in this, but the point is to alert you quickly to potential misunderstanding. As I said, I don't recall any previous opinion that Prolog's reliance on recursion was a weakness to be overcome. If I mistook your point (aka "have no clue"), please enlighten me.

```
>> As you experimented, you found that asserting
>> dynamic facts does not preserve the semantics
>> of variable identification in the same way as
>> their unification in rule chaining.  That's a
>> keen observation on your part, and I look
>> forward to seeing if your opinion of recursion
>> (and polymorphism??) improves with practice.
>
> Same question. Or maybe you are just pulling my leg.
```

I'm not pulling your leg. It seems to me you were quite diligent to discover by testing (using varied versions) that your implementation of reverse doesn't preserve the ordinary semantics of unification.

To me your implementation is "swatting a fly with a sledgehammer". Its dependence on non-local state could be improved by retracting all was_in_list/1 facts at the outset, either in addition to or in preference to the retraction at the end.

Here's the usual Prolog implementation of reverse/2 by "embedding" it in an auxiliary version reverse/3:

```
reverse(A,Z) :- reverse(A,[],Z).

reverse([],Z,Z).
reverse([H|T],X,Z) :- reverse(T,[H|X],Z).
```

This seems to me a more self-contained implementation. Of course if yours could be shown to have a better performance, then I'd be happy to grant that point.

```
>> There are generalizations to the cut construction
>> in various implementation dependent ways.  One keyword
>> to search on is "getBackTrack" (resp. cutBackTrack),
>> which are useful in implementing Prolog within
>> Prolog, among other things.
>
> Those are quite unimaginative and known in various
implementations, e.g.,
> in XSB they are named '_$savecp'/1 and '_$cutto'/1, and I
could imagine
> that any Prolog implementation must have those, even if
they are not in
> the manual. I was thinking about something more far
reaching than that.
>
>> I'm not sure what you mean by saying the standard
version
>> of member/2 is "both left and right recursive."  Here:
>
>> member(H,[H|_]).
>> member(X,[_|T]) :- member(X,T).
>
>> This is "tail recursive" in the sense that when
>> the outer instance of the member/2 goal invokes
>> the inner instance, it's the final subgoal and
>> there are no "open" backtrack points to consider.
>> So it can be implemented by the Prolog engine in
>> an efficient manner.  What's not to like?
>
```

> It seems you do not know what I mean by left- and right-recursive.
> Maybe I should rephrase that to head- and tail-recursive.

I don't know what head-recursive should mean. In the present application, the clause which directly relates to head H does not involve recursion (a predicate that calls itself).

> What is an "open" backtrack point ?
> Do there exist "closed" backtrack points as well then ?

By backtrack point I mean a goal or subgoal for which there are alternative ways of succeeding/satisfaction.

One may certainly have a case in which all but one of the possibilities have been tried and failed, so that only one possibility of success remains. In this way what once may have been an "open" backtrack point (ie. multiple alternatives to explore) can convert into a "closed" backtrack point. I don't insist on this terminology, though the distinction in states is certainly important enough to deserve a definition. Choice point is another phrase denoting places in the rule chaining where alternatives still exist. Deterministic is a word used to describe predicates that succeed in at most one way, while nondeterministic denotes predicates which might provide more than one solution.

> You seem to be an expert in Prolog implementation: what can the
> Prolog engine (PE ?) do so efficiently for tail-recursive calls it cannot
> do for others ?

Loosely speaking there is an optimization for such tail-recursive predicates as I've described that allows the implementation by recursion to be reduced to an implementation by iteration/looping. This saves stack space, which can be critical if the recursion goes deep, and can boost the speed of computation.

> I also find the terminology "inner and outer instance of a
> goal" quite strange: no Prolog textbook I have read mentions it.

It fairly common across many languages to refer to the scope of variables in a calling function/predicate as "outer" and to the scope in a called function/predicate as "inner". For recursion we have, at least indirectly, a function/predicate that calls (invokes) itself. Feel free to suggest a better vocabulary for drawing this distinction if inner/outer lacks clarity.

> Now all the above is just chit-chat: can we focus on the real
> contribution of my original post ? The database stuff, the design

```
> patterns, the language design advice ? Please, I do not
want to stall !
```

best wishes, chip

From: **A.L.**
Subject: Advice from a newbie

anyway <b@rt.luna.be> wrote:

```
>Dear comp.lang.prolog,
>
>Sorry this is long, but I really have important things to
show ...
>
>I recently got my hands on Prolog. I really think I got
the hang of
>it: I can do anything I used to do in my (other) most
favourite
>languages, and often the Prolog version is even more
concise, although
>probably somewhat obscure to the non-informed. And to my
surprise
>(well, actually not really :-), I have come up with
versions of common
>predicates that apparently nobody conceived of before:
here is my
>version of the reverse predicate - I named it rev/2
because it could
>otherwise clash with something in the library:
>
>rev(List,_) :-
>        member(X,List),
>        asserta(was_in_list(X)),
>        fail.
>rev(_,RevList) :-
>        findall(X,retract(was_in_list(X)),RevList).
```

Few threads above (thread name "append problem") there was discussion regarding assert/retract. Although I don't agree with all opinions expressed there, the following statement issued by Matthew Huntbach seems to be written directly for you.

Quote:
"If you have to use asssert/retract to simulate mutable variables, it indicates you really haven't started thinking in Prolog. I only occasionally glance at this

newsgroup, but over the years, many times, I've seen Prolog newbies here struggling and jumping to using assert/retract where there are much more elegant ways of doing what they want to do which don't use them.

Pure logic and functional languages just don't use mutable variables. In Prolog, assert/retract are rather ugly things mainly used to get round the fact that otherwise no information is saved over backtracking. There are specialist cases where the self-modifying code they provide is useful - but this is not something newbies should be concerned with, and it has no place in an introductory tutorial.

Matthew Huntbach"

From: **A.L.**
Subject: Advice from a newbie

anyway <b@rt.luna.be> wrote:

```
>I do have some more advice for the Prolog community, in
particular
>about the cut (!/0) whose scope I find rather limited, but
maybe more
>about that later: this was enough for one day's work. I am
off now, to
>refactor some more predicates. On my todo list are the
inverse of
>merg/3, and I would really like a version of member/2
which does not
>come from the library. I have seen the library version of
member/2 and
>it is extremely recursive: it is both left and right
>recursive. Probably Prolog programmers don't even know
this, or else
>they would avoid it as the plague: how can you be sure it
works
>correctly ? It must be based on an weird programming
paradigm from
>the seventies or maybe even the sixties. Anyway, if you
feel you can
>contribute, please do so in this newsgroup: together we
can improve the
>world, or at least Prolog !
```

Good jokes! With good advices wait unil you get rid of "newbie" status.

See you 5 years from now.

From: **anyway <b@rt.luna.be>**
Subject: Advice from a newbie

A.L wrote:
```
> Good jokes! With good advices wait unil you get rid of
"newbie"
> status.
>
> See you 5 years from now.
```

It seems that the established Prolog masters are not happy with what I wrote and propose. Maybe they could first tell me: have they ever seen versions of append, reverse or merge that ressemble even remotely what I wrote ? If not, can they judge them on their merit instead of on my newbie status ? Probably too much to ask. Anyway, here comes my proposal for a new cut device.

There are basically two ways to cut alternatives in Prolog: once/1 and the cut (!/0) [I will treat if-then-else at some later occasion perhaps - needless to say, there is something wrong there as well].

Both are limited in their scope: once(G) only cuts alternatives in G, and the cut only EVERYTHING to the left. Lee Naish - a very enlightend person indeed - once proposed a cutting predicate (whose name I forgot) that would only take away the alteratives of the head: it didn't make it into ISO (probably too far ahead of its time). The predicates that implementations hide for users - like '_$savecp'/1 and '_$cutto'/1 in XSB - are also quite limited: the scope is from the save up to the cutto.

Here is what I want (actually badly need almost anytime I write a larger piece of Prolog - something I am now doing on a regular basis). I am illustrating it with an example, but I am using variables only where they matter for showing my point:

```
p :- identify_goal(q,Id), r, foo(Id).

  foo(Id) :- s , cut_goal(Id), ....
```

Two new builtin constructs are used: identify_scope/2 executes its first argument (like call/1), and on success unifies its second argument with an identifier of the goal. cut_goal/1 cuts all choices left by the execution of the goal identified by its argument.

Note that the identifier can be passed around as any other Prolog object.

Note that the choices for the intermediate goals (r,foo,s) are not affected.

It should be clear that those two builtins can be used to implement once/1, the cut, Lee Naish's head cut, the if-then-else and even the if/3 (as in SICStus Prolog) - but the latter needs some inventivity (anyone wants to try it ?).

If you would like to comment on these two new builtins, please do so, whether on their usefulness, their operational semantics, how difficult it would be to implement them.

With some hesitation - because I am not absolutely sure that this is useful, but at least it is thinkable - I am also showing you another quite innovative related predicate: uncut_goal/1. Again an example:

```
p :- identify_goal(q,Id), r, !, foo(Id).

 foo(Id) :- s , uncut_goal(Id), ....
```

Note the differences: there is a cut after r, and foo/1 uses uncut_goal. The idea is simple and intuitive: the cut has removed the choices from q. Uncut_goal RE-INSTALLES these cut away choices !

I think this might turn out very powerfull as a programming aid: local decisions to cut away choices, can be undone elsewhere. This shows a similarity with undoing bindings to variables on backtracking, and I must admit, I haven't worked out the analogy, but I anticipate a duality here that might be best expressed in terms of categories. If someone feels up to it: please go ahead.

From: **A.L.**
Subject: Advice from a newbie

anyway <b@rt.luna.be> wrote:

```
>It seems that the established Prolog masters are not happy
with what I
>wrote and propose.
```

No.

```
>Maybe they could first tell me: have they ever seen
>versions of append, reverse or merge that ressemble even
remotely what
>I wrote ?
```

Fortunately, no.

```
> If not, can they judge them on their merit instead of on
my
>newbie status ? Probably too much to ask. Anyway, here
comes my
>proposal for a new cut device.
>
```

Study, study, study! There will be never enough.

From: **Matthew Huntbach**
Subject: Advice from a newbie

A.L. wrote:
```
> Few threads above (thread name "append problem") there
was
> discussion regarding assert/retract. Although I don't
agree with all
> opinions expressed there, the following statement issued
by Matthew
> Huntbach seems to be written directly for you.
>
> Quote:
>
> "If you have to use asssert/retract to simulate mutable
variables,
> it indicates you really haven't started thinking in
Prolog. I only
> occasionally glance at this newsgroup, but over the
years, many
> times, I've seen Prolog newbies here struggling and
jumping to using
> assert/retract where there are much more elegant ways of
doing what
> they want to do which don't use them.
>
> Pure logic and functional languages just don't use
mutable
> variables. In Prolog, assert/retract are rather ugly
things mainly
> used to get round the fact that otherwise no information
is saved
> over backtracking. There are specialist cases where the
> self-modifying code they provide is useful - but this is
not
> something newbies should be concerned with, and it has no
place in
> an introductory tutorial.
```

Indeed, I think this poster well illustrates my point.

The solution proposed uses the clause database as a mutable list, and implements loops over it using fail and findall. Well, yes, you *can* do this, but the point is why? The only reason given by this poster is to avoid recursion. Again, why? There are very simple and obvious ways of solving this problem

using recursion, there is no need to go to the complexity and inefficiency of changing the clause database and simulating loops using fail and findall to solve it.

So it looks to me as if this person, rather than embracing the clear declarative way of solving problems, has instead used the crutch of assert/retract to solve it in a way that is still thinking in terms of loops over mutable variables. This person needs to be taught to get over his or her fear of recursion and to be able to use it as a natural tool in the programmer's toolkit. The whole point of logic programming is that one can state a solution in terms of simple relationships, but that does often involve recursion. If one isn't doing that, then one is using Prolog, but not doing logic programming. In that case, why use Prolog?

From: **Jan Wielemaker**
Subject: Advice from a newbie

anyway <b@rt.luna.be> wrote:

```
> Here is what I want (actually badly need almost anytime I
write a
> larger piece of Prolog - something I am now doing on a
regular basis).
```

Its a bit odd that you appear to be one of the few that needs this :-) Otherwise, it looks most like goto *var of gcc. Something that allows you to write very unreadable code.

```
> I am illustrating it with an example, but I am using
variables only
> where they matter for showing my point:
>
>       p :- identify_goal(q,Id), r, foo(Id).
>
>       foo(Id) :- s , cut_goal(Id), ....

> With some hesitation - because I am not absolutely sure
that this is
> useful, but at least it is thinkable - I am also showing
you another
> quite innovative related predicate: uncut_goal/1. Again
an example:
>
>       p :- identify_goal(q,Id), r, !, foo(Id).
>
>       foo(Id) :- s , uncut_goal(Id), ....
```

You'll definitely make friends with the memory vendors. One of the big reasons for having choicepoint pruning is to allow the system to remove a lot of junk. If it

needs to be able to get this junk back from the wastebin it no longer can do this.

```
> My blessings,
>
> b@rt.luna.be
```

Luna.be? ... :-)

From: **Pinapple**
Subject: Advice from a newbie

Matthew Huntbach wrote:

```
> There are very simple and obvious
> ways of solving this problem using recursion...
```

I don't really disagree with the "general gist" of what you have to say here, or elsewhere, and your points are well-taken. But to nitpick a small thing, I have noticed you use the phrases "simple" and "obvious" several times (in previous posts) to describe a potential way to solve a problem (your way), vs. another way (the "newbie way," or "unprolog way"). I think it should occur to you that if it were as "simple" and "obvious" as you say it is, all these newbies would be doing it your way, don't you think? The point simply is, "simple" and "obvious" are *OBVIOUSLY* relative terms, and you shouldn't use them as if they aren't. You remind me of Kasparov wondering why the chess novice can't see the "simple" and "obvious" mate in 15 that he so easily sees. His ability to see it is what makes him an expert. Your ability to see these Prolog things is what makes you (and Demoen and Triska and Wielemaker, etc.) an expert.

It must also be said that Prolog in many cases forces one to do extreme backflips in order to do something truly simple and obvious, that would take one line of code in a more traditional language. Reminds me of programming in a stack language and being forced to do all manners of pops, dups, rolls, dips, etc. just to get access to the value I need to manipulate (and then I have to do it all over again for the next value to manipulate). The stack experts say "What's the problem? The simple and obvious solution here is a 3-point roll-reverse-dup-dip-swap maneuver, combined with a monadic pick-rotate- pop!" And I'm like "Uh... in another lanugage I could simply write x=x +1."

From: **Matthew Huntbach**
Subject: Advice from a newbie

pineapple wrote:

```
> Matthew Huntbach wrote:
>> There are very simple and obvious
>> ways of solving this problem using recursion...
>
```

> I don't really disagree with the "general gist" of what you have to
> say here, or elsewhere, and your points are well-taken. But to
> nitpick a small thing, I have noticed you use the phrases "simple" and
> "obvious" several times (in previous posts) to describe a potential
> way to solve a problem (your way), vs. another way (the "newbie way,"
> or "unprolog way"). I think it should occur to you that if it were as
> "simple" and "obvious" as you say it is, all these newbies would be
> doing it your way, don't you think? The point simply is, "simple" and
> "obvious" are *OBVIOUSLY* relative terms, and you shouldn't use them
> as if they aren't. You remind me of Kasparov wondering why the chess
> novice can't see the "simple" and "obvious" mate in 15 that he so
> easily sees. His ability to see it is what makes him an expert. Your
> ability to see these Prolog things is what makes you (and Demoen and
> Triska and Wielemaker, etc.) an expert.

I don't think the very basic use of recursion which is required to write reverse or append in Prolog is equivalent to looking 15 moves ahead in chess. To me, doing it this way is both simple and obvious because it uses only the most basic core aspect of Prolog, it can be written in a few lines, and ONCE ONE HAS GOT USED TO IT is so intuitive that one hardly has to think about it.

The words in capitals are the key. I appreciate that, for some reason, these techniques require a bit of getting used to. For some reason, many newbies find recursion hard to use, and will go to great lengths using complex side-aspects of the language, to avoid it. I am not fully sure why that is, perhaps you and the orignator of the thread can say why.

Let us consider appending two lists, as one of the examples considered, just about the most basic piece of recursion you can get in Prolog.

To append two lists, if the first is empty the result is the same as the second.

Otherwise, to append two lists, if you append everything except the first item of the first list to the second, then append the first item of the first list to the result, you get the append of the two lists.

This is English, not Prolog, but it translates almost directly to the two Prolog

clauses of standard append. Now, you and the originator of this thread clearly do regard this as complex, to the point of likening it to a chess grandmaster looking 15 moves ahead and seeking instead to use far more complex programs which involve self-modifying code. Why?

From: **A.L.**
Subject: Advice from a newbie

pineapple wrote:

```
>> There are very simple and obvious
>> ways of solving this problem using recursion...
>
> I don't really disagree with the "general gist" of what
you have to
> say here, or elsewhere, and your points are well-taken.
But to
> nitpick a small thing, I have noticed you use the phrases
"simple" and
> "obvious" several times (in previous posts) to describe a
potential
> way to solve a problem (your way), vs. another way (the
"newbie way,"
> or "unprolog way").  I think it should occur to you that
if it were as
> "simple" and "obvious" as you say it is, all these
newbies would be
> doing it your way, don't you think?  The point simply is,
"simple" and
> "obvious" are *OBVIOUSLY* relative terms, and you
shouldn't use them
> as if they aren't.
```

"I don't understand Relativity Theory and I don't know Relativity Theory. But I am sure that Einstein was wrong. Therefore, I present my own Theory of Everything, only 1 page long"..

Quote from sci.physics.

From: **Jan Wielemaker**
Subject: Advice from a newbie

Matthew Huntbach wrote:

```
> To append two lists, if the first is empty the result is
the same as the
> second.
>
```

```
> Otherwise, to append two lists, if you append everything
except the first
> item of the first list to the second, then append the
first item of the
> first list to the result, you get the append of the two
lists.
>
> This is English, not Prolog, but it translates almost
directly to the two
```

But there are many more ways to describe append that do not translate that well into Prolog. What about

> To append two lists, create a new list, put all elements of the first into the new list, then put all elements of the second into the new list.

English that translates into Prolog is English that does not use some notion of an imperative variable and splits list into the head (an element) and tail (a list). With different but similar subsets of the English language you can describe any program in any language in English.

I'm convinced the tradional Prolog way of defining predicates that transform datastructures is, once mastered, a powerful mechanism that is less prone to errors than imperative alternatives. I doubt it is more natural to the average person (whatever that may mean).

From: **Matthew Huntbach**
Subject: Advice from a newbie

I'm not saying it is more natural. It's simple, in that it can be written down very concisely in English and that translates directly to Prolog, and it's obvious ONCE YOU GET USED TO THINKING THAT WAY. I think it plain, however, from the way that many newbies struggle with it, that there is a conceptual barrier which many people find difficult to cross. That is why I think, when learning Prolog, it is important to do plenty of exercises in the basic recursion and backtracking way to ensure thorough familiarity with that, rather than be given the crutch of assert/retract which enables people to escape into an imperative world. If you'd rather be programming in an imperative way, then why use Prolog in the first place?

This issue goes right back to the origins of Prolog, when it was genuinely felt that it was so simple and natural that using it would make programming so much easier. When this seemed not to be the case, the first Prolog people wondered if the problem was that somehow their students' brains had been damaged by doing imperative programming first, and thought that maybe if they were taught

Prolog first they would naturally think that way. But even there, it does seem that somehow there are a small number of people who find recursion simple and natural, and a large number who don't. I'm still optimistic enough to think the barrier can be jumped with practice, rather than to accept that most of humanity just can't and never will grasp it. But if that really is the case, we might as well give up on Prolog.

From: **Jan Wielemaker**
Subject:Advice from a newbie

Matthew Huntbach wrote:

```
> a conceptual barrier which many people find difficult to
cross. That is
> why I think, when learning Prolog, it is important to do
plenty of
> exercises in the basic recursion and backtracking way to
ensure thorough
> familiarity with that, rather than be given the crutch of
assert/retract
> which enables people to escape into an imperative world.
If you'd rather
> be programming in an imperative way, then why use Prolog
in the first
> place?
```

Very few people will oppose this view. The power of Prolog is recursion, logical variables and backtracking. Any course must teach these and avoid as much as possible explaining there are ways to to imperative programming in Prolog after all. Even with all tricks, Prolog is a lousy imperative programming language!

```
> rather than to accept that most of humanity just can't
and never
> will grasp it.
> But if that really is the case, we might as well give up
on Prolog.
```

Not so soon. I know enough people that have fun and make a living because they understand these mechanisms :-) I very much doubt there is any hope Prolog (or more generally Logic Programming) will reach wide acceptance ever though. Except for mathematicians, its just not natural enough ...

From: **Pinapple**
Subject: Advice from a newbie

Matthew Huntbach wrote:
```
> I don't think the very basic use of recursion which is
required to
> write reverse or append in Prolog is equivalent to
looking 15 moves ahead
> in chess.
```

I don't either, personally, but that wasn't the point. The point was, "simple" and "obvious" are obviously relative terms. If they aren't relative, then you seem to imply that newbies purposefully do the non-simple and non-obvious even though they see the simple and obvious solution staring them in the face.

From: **anyway <b@rt.luna.be>**
Subject: Advice from a newbie

A.L wrote:
```
>> If not, can they judge them on their merit instead of on
my
>> newbie status ? Probably too much to ask. Anyway, here
comes my
>> proposal for a new cut device.
>
> Study, study, study! There will be never enough.
```

Study is best performed under the guidance of masters. But apparently, the Prolog masters can only say "study" or "get used to it" - that's even worse than Zen masters who at least have a deeper message in their otherwise incomprehensible words - even if that message misses the point as well.

Anyway, I am not yet giving up on you guys !

Why is recursion a bad thing ? Here is the answer: remember the old days when things were better ? One of those better things was ... the absence of recursion in languages like Fortran and Cobol. Most new Prolog programmers these days are not old enough to remember that of course, but the abhorrence from recursion is deeply grafted in our DNA: it is just plain wrong to define a concept in terms of itself, this is clear to any person in the street, whether programmer, matematician, welder or book keeper. The only way to make sense of such self-defining definitions is to rely on a meta-principle: induction. But induction does not always apply: you must have a well-founded order to do it on, and you must check that your recursive definition respects that order. That's why textbooks give a definition of Ackerman's function and imediately ask you to prove that this actually defines something. Now, it is clear that proving anything at all, goes way beyond most programmer's skills. Still, without proofs, all of your recursively defined predicates may just mean gibberish !

On the other hand, even babies use iteration: cry until you get food. No nonsense like "I cry and if I do not get food, I will call the same procedure again".

But the focus of all this is wrong: my proposals for new builtins (or a new semantics for old ones), my new design patterns for Prolog ... no fundamental reaction to that. Just something from Mister SWI about memory vendors - I might react to that in a later contribution.

I am sure of what will happen: the gurus will put me in their kill-file, the other newbies with great ideas will be put off by the lack of response to novelty in this news group and the Prolog masters will keep on helping students out with their homework. I might just move on to another newsgroup: that's not a threath, it is just a real possibility. But sad for Prolog. And I find the reaction by A.L. (Einstein, relativity theory ...) all the more inappropriate, because also my PhD supervisor was convinced that general relativity was too complicated to be correct.

So, please can someone with brains come to the real point ?

From: **Pinapple**
Subject: Advice from a newbie

Matthew Huntbach wrote:

```
> Now, you and the originator of this thread
> clearly do regard this as complex, to the point of
likening it to a chess
> grandmaster looking 15 moves ahead and seeking instead to
use far more
> complex programs which involve self-modifying code. Why?
```

Funny you should ask. The solution you are referring to where I used "self-modifying code" was actually an attempt to get *TO* the basic Prolog ability of backtracking (something you say newbies should be drilled on - and I agree). To me, the other solutions were more exotic and fancy, and required much more knowledge of Prolog libraries and built-in predicates. I was just trying to stick to ultra-basics. I know you disagree, and think that what I did was non-obvious and uber-complex, but I suppose that's what makes opinions what they are - opinions.

I was simply trying to get to a point where I could do a simple "fact1(X), fact2(Y), fact3(Z), fail" and have Prolog backtrack and spit out all the solutions. In that, I was a success, and I call that about as fervent of an attempt to get at the "basics" as you'll find. Please - criticize my solution all you want. I beg you to. Just understand that my attempt was actually to try to do "standard, ultra- basic" Prolog. If I failed in that attempt, I suppose it's better to have tried and failed than to not have tried at all.

From: **Pinapple**
Subject: Advice from a newbie

anyway <b@rt.luna.be> wrote:
```
> but the abhorrence from recursion is deeply grafted in
our DNA: it is just
> plain wrong to define a concept in terms of itself....

> On the other hand, even babies use iteration:
```

Huntchback, I think this guy actually has a good point. In peoples' everyday lives, they don't think recursively, they think iteratively. If some guy on the street asks me directions to go somewhere, I don't give him a recursive set of directions. I give him an iterative set of directions. When I work on a car engine, I don't look at a recursive instruction manual, I look at an iterative manual. For just about every single problem I can think of in daily life - even mowing my lawn - I don't think about the problem recursively...

```
mowlawn_finished([]).
mowlawn-finshed([H|T] :- cutgrass(H), mowlawn_finished(T).
```

I think about it iteratively (and so does everyone else). I think about it in terms of doing strips at a time, making quarter turns or 180 degree turns with the mower, checking to see if there is more grass left to cut, etc. Basically a big while loop with a bunch of instructions and checks in the middle of it.

Note that I am not putting down recursion, nor the need for a newbie to learn recursion in Prolog. Nor am I advocating Prolog to (necessarily) get away from recursion. I am merely answering your previous question by saying I believe this guy makes a strong point about it "being in the DNA." Nobody walks around thinking recursively in their daily lives, in solving daily problems like changing the baby's diaper or making food. At least, I don't know anyone who does.

From: **A.L.**
Subject: Advice from a newbie

anyway <b@rt.luna.be> wrote:
```
>But the focus of all this is wrong: my proposals for new
builtins (or a
>new semantics for old ones), my new design patterns for
Prolog ... no
>fundamental reaction to that. Just something from Mister
SWI about memory
>vendors - I might react to that in a later contribution.
```

There will be NO reaction - for the same reeason why on sci.physics there is no reaction to discoveries of perpetual motion.

```
>I am sure of what will happen: the gurus will put me in
their kill-file,
>the other newbies with great ideas will be put off by the
lack of
>response to novelty in this news group and the Prolog
masters will keep on
>helping students out with their homework. I might just
move on to another
>newsgroup: that's not a threath, it is just a real
possibility.
```

Good! Please!

From: **Matthew Huntbach**
Subject: Advice from a newbie

Jan Wielemaker wrote:

```
> Matthew Huntbach wrote:
>> rather than to accept that most of humanity just can't
and never will
>> grasp it. But if that really is the case, we might as
well give up on
>> Prolog.

> Not so soon. I know enough people that have fun and make
a living
> because they understand these mechanisms :-) I very much
doubt there is
> any hope Prolog (or more generally Logic Programming)
will reach wide
> acceptance ever though. Except for mathematicians, its
just not natural
> enough ...
```

When Prolog was first developed, the main argument for it was that as it was based on a human mechanism, logic, rather than the electronics of a computer as standard programming languages are, it should be much more natural and easy to use than standard programming languages. So if it isn't, what's the point?

From: **Matthew Huntbach**
Subject: Advice from a newbie

anyway <b@rt.luna.be> wrote:
> Why is recursion a bad thing ? Here is the answer:
remember the old days
> when things were better ? One of those better things
was ... the absence
> of recursion in languages like Fortran and Cobol. Most
new Prolog
> programmers these days are not old enough to remember
that of course,

In my case no, when I was first taught Prolog I found it fun and so much more easy to use than the imperative languages of the day. I didn't find recursion a barrier, it seemed to me to be a natural way of expressing solutions to problems.

> but the abhorrence from recursion is deeply grafted in
our DNA: it is just
> plain wrong to define a concept in terms of itself, this
is clear to any
> person in the street, whether programmer, matematician,
welder or book
> keeper. The only way to make sense of such self-defining
definitions is
> to rely on a meta-principle: induction. But induction
does not
> always apply: you must have a well-founded order to do it
on, and you must
> check that your recursive definition respects that order.
That's why
> textbooks give a definition of Ackerman's function and
imediately ask you
> to prove that this actually defines something. Now, it is
clear that
> proving anything at all, goes way beyond most
programmer's skills. Still,
> without proofs, all of your recursively defined
predicates may just mean
> gibberish !

If you insist, recursion can be described in terms of how it's implemented underneath, which is iteratively putting things onto and taking them off stacks. But being able to use recursion just opens up problem solving techniques - to suggest it's just "plain wrong" is mad - it's used in all sorts of common software.

Look again at my recursive description of list appending:

"To append two lists, if the first is empty the result is the same as the second. Otherwise, to append two lists, if you append everything except the first item of the first list to the second, then append the first item of the first list to the result, you get the append of the two lists."

Just what is wrong with that? Why do you suggest there is some natural "abhorrence" of it that you recoil from it?

Or what about

"to sort a list if it has one or no elements it is sorted, otherwise divide it into two lists, all those elements less than the first, all those greater than the first, sort each list, append the sorted lists together"

This is quick sort, if you will not use recursion, you will be stuck with less efficient sorts. Can you point out exactly what is so abhorrent about it that you would prefer to boycott it than try to understand it?

From: **Matthew Huntbach**
Subject:Advice from a newbie

pineapple wrote:

```
>> but the abhorrence from recursion is deeply grafted in
our DNA: it is just
>> plain wrong to define a concept in terms of itself....
>
>> On the other hand, even babies use iteration:

> Huntchback, I think this guy actually has a good point.
In peoples'
> everyday lives, they don't think recursively, they think
iteratively.
> If some guy on the street asks me directions to go
somewhere,
```

If I want to go somewhere, and I don't know the way, I think of a place in between where I am now, and that other place. Then either the journey from that in between place to one of the other places is a straight road, or I have to work out the journey from my starting point to the mid-point, and work out the journey from th mid-point to where I want to go. That's recursion.

```
> Note that I am not putting down recursion, nor the need
for a newbie
> to learn recursion in Prolog.  Nor am I advocating Prolog
to
> (necessarily) get away from recursion.  I am merely
answering your
> previous question by saying I believe this guy makes a
strong point
> about it "being in the DNA."  Nobody walks around
thinking recursively
> in their daily lives, in solving daily problems like
changing the
```

> baby's diaper or making food. At least, I don't know anyone who does.

Suppose I'm climbing to the top of the building. I climb a set of steps, turn round and either I'm at the top, or there's another set of steps facing me. If there's another set of steps facing me, I've reduced the number of sets of steps I have to climb by one, but I'm still faced with a smaller version of the same problem - climbing a number of sets of steps to the top of the building. That's recursion.

If I'm mowing the lawn, what do I do - I mow part of the lawn, then I've reduced the problem to mowing the rest of the lawn, which I do in the same way. That's recursion.

Now, I have to accept, because I've seen it so many times, and this guy is a particularly pathological case, that many people do find recursion hard to use, particularly when expressed using formal notation. But I think one of the points about Prolog is that it's a language where recursion is even more central than other languages, because it replaces loops. That's why it can be a good teaching tool, because it *forces* people to use recursion rather than seek escape routes from it. I don't think it helps just to write it off as so difficult one won't even bother with it, and instead will use all sorts of complicated mechanismss to get round it.

From: **Boris Borcic**
Subject:Advice from a newbie

anyway <b@rt.luna.be> wrote:
> Why is recursion a bad thing ? Here is the answer:
remember the old days
> when things were better ? One of those better things
was ... the absence
> of recursion in languages like Fortran and Cobol.

Actually the first time I heard the name of recursion was in the mid-70es when I complained to the optional "informatics" teacher that I couldn't translate into working Fortran the determinant computation method I had just learned in math class.

"Fortran isn't recursive" I was told. Later that year I wrote a "random instantiator for formal grammars" that involved encoding (recursive) grammars as Fortran statements, then having the functions invoked by the statements, rewrite the unoptimized (and thus very regular) machine code around their calling points, to turn said machine code into a graph representation of the grammar that was finally handed to a small interpretor.

That time the mystified teachers exclaimed "but Fortran isn't recursive !??..."

> Most new Prolog

> programmers these days are not old enough to remember that of course, but
> the abhorrence from recursion is deeply grafted in our DNA:

Thank God for biodiversity :) And grammar :)

From: **Jan Wielemaker**
Subject:Advice from a newbie

Matthew Huntbach wrote:
> Jan Wielemaker wrote:
>> Matthew Huntbach wrote:
>
>>> rather than to accept that most of humanity just can't and never will
>>> grasp it. But if that really is the case, we might as well give up on
>>> Prolog.
>
>> Not so soon. I know enough people that have fun and make a living
>> because they understand these mechanisms :-) I very much doubt there is
>> any hope Prolog (or more generally Logic Programming) will reach wide
>> acceptance ever though. Except for mathematicians, its just not natural
>> enough ...
>
> When Prolog was first developed, the main argument for it was that as it
> was based on a human mechanism, logic, rather than the electronics of
> a computer as standard programming languages are, it should be much more
> natural and easy to use than standard programming languages. So if it
> isn't, what's the point?

In good computer science tradition however I'm afraid nobody actually performed the tests. Correct me if I'm wrong. In social science they would take a lot of students, split them into two groups and run an experiment. Thats not an easy experiment as it depends on how you prepare the students, the problems and the metrics you define for a good/better result. Still, if this experiment was carried out in enough CS departments we could probably say something sensible about this claim.

CS has developed a lot of languages, but basically no evaluation :-) They are (almost) all equally expressive (= Turing complete). Some might not be able to represent certain problems with optimal complexity. Some programmers like logic, others iteration, some like recursion, some don't, some like types, some don't, some like objects (in many flavours) and some don't.

As we cannot compare languages the only remaining question is whether you like the language or not. I happen to like Prolog :-) As long as there are enough people like me, Prolog has a point. Its no different than pizas, as long as there enough people that like them, making Pizas makes sense.

B.t.w. it is totally irrelavant to this discussion what the original developers had in mind.

From: **Pineapple**
Subject:Advice from a newbie

Matthew Huntbach wrote:

```
> When Prolog was first developed, the main argument for it
was that as it
> was based on a human mechanism, logic, rather than the
electronics of
> a computer as standard programming languages are, it
should be much more
> natural and easy to use than standard programming
languages. So if it
> isn't, what's the point?
```

Good question. The answer is, there doesn't necessarily need to be a point. It's a language, just like any other. If there are people who program in it and like it - great. If there aren't, there aren't. It's as simple as that.

From: **Pineapple**
Subject: Advice from a newbie

Matthew Huntbach wrote:

```
> If I want to go somewhere, and I don't know the way, I
think of a place
> in between where I am now, and that other place....
> That's recursion.
```

You're an alien :-)

```
> I don't think it
> helps just to write it off as so difficult one won't even
```

```
bother with it,
> and instead will use all sorts of complicated mechanismss
to get round it.
```

Agreed.

From: **Morin Thurgh**
Subject: Advice from a newbie

I feel myself a bit trolled in writing this, both because I am not a Prolog Master, and because the discussion has already degenerated, but I'll give it my 2 cents nonetheless.

It seems to me that the basic issue in your writing is that you expect Prolog underneath model of computation to be different from what it is: the database alteration by means of retract and assert* is not intended to be particularly efficient, because the focus of Prolog approach is meant to be more on rules application than on facts alteration.

Modifying the knowledge base just to store temporary results is obviously possible, and as other threads would tell you, exactly one of the things that newbies, like you qualify yourself, tend to overuse.

I saw, long time before ISO, code similar to yours in many students code snippets; they were OK, but not as elegant as other solutions designed not to leave intermediate traces of their execution into the knowledge base they then had to remove right before completion (that many times were not fully cleaned too, defect that does not seem to affect your examples). This is to plainly say that your code looks correct, but neither particularly innovative, nor actually efficient.

The lack of efficiency in altering the knowledge base loose relevance when it can help implementing something like non-chronological backtracking, but that's a different story.

anyway <b@rt.luna.be> wrote:

```
> 1) maybe you find my expercience too limited, but my
versions of rev
> and app show clearly that the sequence member(X,L),
asserta(foo(X))
> is very useful: it is worth lifting this to the status of
a design
> pattern and have it abbreviated to something like FORALL
X IN L
> ASSERT foo(X) What do people think about this ? [I am
transferring
> stuff from my other favourite languages - in particular
COBOL - to
> Prolog here, but this will help wider acceptance of
```

```
Prolog, I am
> sure]
```

[Well, I'm afraid statements like your last sentence above are not preparing the right mood around your other text! :) Seriously: I know almost nothing about Object Oriented Cobol, but verbosity, rigidity of structure, and imperativeness of plain ol' COBOL have quite a little to share with Prolog, IMHO.]

I personally think that member(X,L), asserta(foo(X)) is already a quite compact way to express your need, not to urge for a more concise replacement.

```
> 2) let's focus on my definition of rev for now (the app
version has
> the same issue): it works fine for lists of atoms, or
integers, mixed
> (atoms and integers) lists and even lists whose elements
are any term
> without variables (some textbooks name these terms
ground, like
> coffee :-); but when I reverse a list with terms with
variables, then
>  these variables become disconnected; here is an example:
>
> ?- rev([X],[Y]), X == Y. No
>
> I have checkd this result in SWI, GNU, XSB, CIAO,
ECLIPSE, SICStus
> and B-Prolog: always the same No.
```

Whether there was not a rev([a],[a])-like fact in your base, this is not surprise. It's a consequence of the closed world hypothesis, combined with the inability of Prolog to perform higher order logic reasoning, I would say.

```
> 3) let's now focus on my app/3: somehow every Prolog
implementation I
>  tried got the complexity of my predicate wrong; my code
is cleary
> O(n) with n the length of the first argument; but I have
noticed that
>  Prolog systems make it into O(n*n) in practice; that's
no good; no
> wonder Prolog is hardly used - please Prolog
implementors, do a
> favour to your language and get this right !
```

Might I ask how you measured that?

```
> 4) both my rev/2 and app/3 exhibit an even stranger
thing: they are
> also linear in the size of the ELEMENTS of the involved
```

lists;
> everyone with half a brain just knows that append and reverse are by
> nature independent of the size of the list elements - so why are
> Prolog implementations getting this one wrong as well ? why is it too
> difficult to get polymorphic types right for you ?

Ditto.

> 5) ...Hard to believe, but there you go ! Apparently, the added call
> to rev/2 operates in the same dynamic predicate space as the toplevel
> call to app/3. For me this almost blew it. Luckily the fix is easy:
> Prolog should encapsulate its asserts/retracts in a dynamicaly scoped
> way. Shouldn't be too difficult for Prolog implementors to
> accomodate for such a nice principle, I'd say.

Here again, the fundamental idea of how the knowledge base is meant to be used is key.

> 6) The following made me really laugh: none of the predicates I wrote
> (including rev/2 and app/3) work backwards, i.e. ?- app (X,Y,[1,2]).
> does not split [1,2] in all possible pairs of lists making up [1,2].
> Still, all Prolog textbooks cannot stop talking about this feature.
>
> It seems that all this fuzz about bidirectional programming in Prolog
> is just a myth. No big deal for me: I will not miss it. But why such
> bad marketing ?

I'm afraid the books you read weren't clear enough on how unification works, or that their authors were cheating.

> I do have some more advice for the Prolog community, in particular
> about the cut (!/0) whose scope I find rather limited, but maybe more
> about that later: this was enough for one day's work. I am off now,

```
> to refactor some more predicates. On my todo list are the
inverse of
> merg/3, and I would really like a version of member/2
which does not
> come from the library. I have seen the library version of
member/2
> and it is extremely recursive: it is both left and right
recursive.
> Probably Prolog programmers don't even know this, or else
they would
> avoid it as the plague: how can you be sure it works
correctly ? It
> must be based on an weird programming paradigm from the
seventies or
> maybe even the sixties. Anyway, if you feel you can
contribute,
> please do so in this newsgroup: together we can improve
the world, or
> at least Prolog !
```

My understanding of right recursion and left recursion relates to grammar production rules and, indirectly, to parser classification, but that doesn't fit your statement. Might I ask you to elaborate further your conjecture on the "weird programming paradigm"?

From: **Matthew Huntbach**
Subject: Advice from a newbie

Morin Thurgh wrote:

```
>> 6) The following made me really laugh: none of the
predicates I wrote
>>  (including rev/2 and app/3) work backwards, i.e. ?- app
(X,Y,[1,2]). does
>> not split [1,2] in all possible pairs of lists making up
[1,2]. Still, all
>> Prolog textbooks cannot stop talking about this feature.
>>
>> It seems that all this fuzz about bidirectional
programming in Prolog
>>  is just a myth. No big deal for me: I will not miss it.
But why such
>>  bad marketing ?

> I'm afraid the books you read weren't clear enough on how
unification works,
> or that their authors were cheating.
```

This is actually a good point - Prolog is often promoted on the grounds that programs can be multi-moded, but in reality its operational model means this is rarely so except for toy examples. Of course, if one insists on a Prolog programming style which eschews Prolog's underlying logic programming mechaniamss and instead concentrates on its extra-logical features, one should be even less surprised to find it doesn't work as advertised.

I think this indicates, again, that a lot of what is said about Prolog now actually goes back go the early optimistic years of logic programming, and does not reflect the limited reality of Prolog as a practical language. That is, there was a logic programming dream of writing statements in pure logic and the machinery underneath would do all the computations and give you answers and you wouldn't have to think about how it did it. Prolog was only a first step towards this, as it wasn't really "magic" and actually was strongly dependent on its underlying mechanism which anyone doing realistic programming in it would need to be thoroughly familiar with. But people confused and confuse now "Prolog" with the logic programming dream, and hence expect Prolog to do what the dream said logic programming would eventually do.

From: **Matthew Huntbach**
Subject:Advice from a newbie

Jan Wielemaker wrote:
Matthew Huntbach wrote:

```
>> When Prolog was first developed, the main argument for
it was that as it
>> was based on a human mechanism, logic, rather than the
electronics of
>> a computer as standard programming languages are, it
should be much more
>> natural and easy to use than standard programming
languages. So if it
>> isn't, what's the point?

> In good computer science tradition however I'm afraid
nobody actually
> performed the tests.  Correct me if I'm wrong.  In social
science they
> would take a lot of students, split them into two groups
and run an
> experiment.  Thats not an easy experiment as it depends
on how you
> prepare the students, the problems and the metrics you
define for a
> good/better result.  Still, if this experiment was
carried out in
> enough CS departments we could probably say something
```

```
sensible about
> this claim.
```

There was some experimental work on this, I'm thinking particularly of Richard Ennals' work on teaching logic programming in schools, which was published in the 1982 Internatinal Conference on Logic Programming.

```
> CS has developed a lot of languages, but basically no
evaluation :-)
> They are (almost) all equally expressive (= Turing
complete).  Some
> might not be able to represent certain problems with
optimal
> complexity.  Some programmers like logic, others
iteration, some
> like recursion, some don't, some like types, some don't,
some like
> objects (in many flavours) and some don't.
>
> As we cannot compare languages the only remaining
question is whether
> you like the language or not.  I happen to like Prolog :-
) As long as
> there are enough people like me, Prolog has a point.  Its
no different
> than pizas, as long as there enough people that like
them, making
> Pizas makes sense.
```

As we have seen from the comments of newbies here, Prolog continues to be promoted on a manifesto which made sense in the early optimistic days of logic programming, but has been disproved by practical experience. That is people are led to believe Prolog is some sort of magic where you don't have to think about what goes on operationally underneath, and are then disappointed to find it doesn't work that way. Also, the claim that it is natural and easy falls down because the natural easy style requires recursion, and as we have seen this really does seem to be a stumbling block to many people.

Prolog has had the advantage of being extensively promoted in academia, though more so in Europe than the US. Very many students have been made to go through learning Prolog. There was at least one university Computer Science department which taught students to program in Prolog first before any other language. To some extent Prolog has survived because there were over-optimistic claims about it when it was first developed, it appealed to academics, and it became a standard part of the university Computer Science curriculum appearing there out of habit rather than because there was any real demonstration of its usefulness. If it were really useful, wouldn't it be more widely used, given that the huge number of people with Computer Science degrees who have gone through learning it, this over a period of nearly three decades now?

I like Prolog, and I've written a book and papers on logic programming, so I can hardly be accused of an irrational dislike or lack of knowledge of it. However, as the continuing stream of puzzled enquiries from newbies in this newsgroup indicates, it does seem there are a lot of people taking it up for no good reason other than they have heard the old propaganda for it, and then find it doesn't live up that propaganda. Also we are constantly finding these newbies saying it is a very difficult language. If the language is difficult, and people are not sure what it is for, and it is only taken up because people have been misled about its capabilities, and few people have ever really discovered domains where it is more suitable than other languages, then just what is its point?

From: **A.L.**
Subject: Advice from a newbie

Matthew Huntbach wrote:

```
>was any real demonstration of its usefulness. If it were
really useful,
>wouldn't it be more widely used, given that the huge
number of people
>with Computer Science degrees who have gone through
learning it, this
>over a period of nearly three decades now?
```

No, because:

a. Majority of programming is GUI development, and Prolog is not strong in this area,
b. Majority of programming is simple data base front end. Prolog is not strong in this area,
c. The rest of programming is graphics, number crunching and such. Prolog is not strong in this area.

In addition, languages don't count that much any more. Important is development environment (IDE) and associated libraries such as libraries for web programming, XML, communication, middleware, interfacing with other languages (what is pain in the butt). Prolog has some libraries, but generally this stuff is 20 years behind the current state of the art.

As for me, the REAL and substantial advantage of Prolog is constraint programming. Doing constraint programming in C++/Java (I have already tried) is like pushing square pegs through round holes. Prolog provides natural environment for constraint programming paradigm.

From: **Matthew Huntbach**
Subject: Advice from a newbie

A.L. wrote:
> Matthew Huntbach wrote:
>> was any real demonstration of its usefulness. If it were really useful,
>> wouldn't it be more widely used, given that the huge number of people
>> with Computer Science degrees who have gone through learning it, this
>> over a period of nearly three decades now?

> No, because:
>
> a. Majority of programming is GUI development, and Prolog is not
> strong in this area,
> b. Majority of programming is simple data base front end. Prolog is
> not strong in this area,
> c. The rest of programming is graphics, number crunching and such.
> Prolog is not strong in this area.

Yes, Prolog is not strong in most of the areas which programming is about.

> In addition, languages don't count that much any more. Important is
> development environment (IDE) and associated libraries such as
> libraries for web programming, XML, communication, middleware,
> interfacing with other languages (what is pain in the butt). Prolog
> has some libraries, but generally this stuff is 20 years behind the
> current state of the art.

Part of the problem, I think, is that Prolog doesn't fit well into the object-oriented model fo programming. Prolog isn't about separate components interacting, it's really about one big global search tree.

> As for me, the REAL and substantial advantage of Prolog is
> constraint programming. Doing constraint programming in C++/Java (I
> have already tried) is like pushing square pegs through round holes.
> Prolog provides natural environment for constraint programming
> paradigm.

Yes, part of the sales pitch for logic programming was that it could involve some complex constraints stuff. As it is, the core of Prolog with its simple search-with-backtracking and unification mechanism is a very crude form of constraint. So if you are saying Prolog is good because it handles constraint programming well, I don't think you're talking about the core of Prolog, rather it's additional facilities added to some varieties of Prolog. But could these equally well be added through appropriate libraries to, say Java? Could you talk me through why not? After all, if it can be done that way, you lose the problem of having to interface with other languages to do the GUI and database front end and number crunching stuff, and you say interfacing with other languages is a pain in the butt anyway. You also lose all those newbies complaining about the language becayse it won't behave in an imperative way.

From: **Pineapple**
Subject:Advice from a newbie

Matthew Huntbach wrote:

```
> You also
> lose all those newbies complaining about the language
becayse it won't
> behave in an imperative way.
```

Maybe Prolog is good, maybe Prolog is bad. It will stand or fall on its own merits, fairly or unfairly. But should a language be structured around newbie complaints, or lack thereof?

From: **A.L.**
Subject: Advice from a newbie

Matthew Huntbach wrote:

```
>Part of the problem, I think, is that Prolog doesn't fit
well into
>the object-oriented model fo programming. Prolog isn't
about
>separate components interacting, it's really about one big
global search tree.
```

Not sure. There was heated discussion here some time ago with mixed results. I am not sure that object orientation and prolog mix well, but now I am developing oposite view... maybe I will write about my experience when I collect more facts.

```
>> As for me, the REAL and substantial advantage of Prolog
is
>> constraint programming. Doing constraint programming in
C++/Java (I
```

```
>> have already tried) is like pushing square pegs through
round holes.
>> Prolog provides natural environment for constraint
programming
>> paradigm.
>
>Yes, part of the sales pitch for logic programming was
that it could involve
>some complex constraints stuff. As it is, the core of
Prolog with its
>simple search-with-backtracking and unification mechanism
is a very crude
>form of constraint. So if you are saying Prolog is good
because it handles
>constraint programming well, I don't think you're talking
about the core of
>Prolog, rather it's additional facilities added to some
varieties of Prolog.
>But could these equally well be added through appropriate
libraries to, say Java?
>Could you talk me through why not?
```

Well... CLP(FD) is pretty well integrated with Prolog; see SICStus or ECLIPSE. It is hard to say that CLP is "add on".

However, the most important issue: constraint programming is declarative. Of course, you can SIMULATE declaraivenes in Java or whatever, but it will be just simulation. Prolog has all what is needed. Declarativenes in optimization and constraint solving was recognized long ago in Operations Research. Languages for defining optimization problems are declarative; see AMPL, OPL, MOSEL, MPL and such. From my own experience: I prototyped some optimization problem in MOSEL. This required about 150 lines of MOSEL. However, for various (technical and business) reasons I had to re-write this model in Java using the library provided by the same vendor that provides MOSEL. It was well over 2000 lines.

I am just completing the project that has 15K lines of Prolog. I am not even trying to think how long it would take to do this in Java (Prolog took about 1000 hours) and how much this would cost.

From: **anyway <b@rt.luna.be>**
Subject: Advice from a newbie

Morin Thurgh wrote:

```
> the database alteration by means of retract and assert*
is not
> intended to be particularly efficient
```

```
[...]
> This is to plainly say that your code looksccorrect, but
neither
> particularly innovative, nor actually efficient.
>
> The lack of efficiency in altering the knowledge base
```

I am glad you raise the efficiency point - which I also did, but I was talking about complexity rather. Isn't efficiency a quality of the implementation of a language, not (necessarily) of the language itself. Or do you imply that assert* and retract must be necessarily inefficient in Prolog ?

```
> > ?- rev([X],[Y]), X == Y. No
> >
> > I have checkd this result in SWI, GNU, XSB, CIAO,
ECLIPSE, SICStus
> > and B-Prolog: always the same No.
>
> Whether there was not a rev([a],[a])-like fact in your
base, this is
> not surprise. It's a consequence of the closed world
hypothesis,
> combined with the inability of Prolog to perform higher
order logic
> reasoning, I would say.
```

First of all, my rev/2 predicate was loaded in the system of course. Secondly, I would be surprised if the explanation of the No is in CWA or the HOL reasoning capabilities of Prolog. Maybe you can explain though ?

```
> Might I ask how you measured that?
[...]
> Ditto.
```

["that" being O(n)] The usual way: repeatedly making the input larger and measuring the time. I know that I cannot prove an asymptotic linear behaviour in this way, but this method works often enough. If you have a proof, that's of course better.

```
> > 5) ...Hard to believe, but there you go ! Apparently,
the added call
> > to rev/2 operates in the same dynamic predicate space
as the toplevel
> >  call to app/3. For me this almost blew it. Luckily the
fix is easy:
> > Prolog should encapsulate its asserts/retracts in a
dynamicaly scoped
> >  way. Shouldn't be too difficult for Prolog
implementors to
```

```
> > accomodate for such a nice principle, I'd say.
>
> Here again, the fundamental idea of how the knowledge
base is meant to
> be used is key.
```

Please, can you explain to me "how the knowledge base is meant to be used" ? This would help me I am sure.

```
> My understanding of right recursion and left recursion
relates to
> grammar production rules and, indirectly, to parser
classification,
> but that doesn't fit your statement. Might I ask you to
elaborate
> further your conjecture on the "weird programming
paradigm"?
```

The notions of left and right recursion carry over directly from grammars to Prolog clauses of course. About "weird programming paradigm": in the beginning there was no recursion and programming could be understood by most people. It still can in languages that do not force you to use recursion, like Java: anybody can program in Java these days, no need to have a PhD in computer science or a twisted mind when mowing the lawn. But languages like Prolog force one to use recursion, or rely on syntactic sugar (like for loops in ECLiPSe) or even worse, you must master things like maplist. All these options obscure what is really going on.

Finally I would like to ask: why are Prolog masters not EXPLAINING things instead of mistifying them when newbies appear in this forum ? Pineapple, the other flea in c.l.prolog's pelt, has complained about that too.

From: **A.L.**
Subject: Advice from a newbie

anyway <b@rt.luna.be> wrote:

```
>Finally I would like to ask: why are Prolog masters not
EXPLAINING
>things instead of mistifying them when newbies appear in
this forum ?
>Pineapple, the other flea in c.l.prolog's pelt, has
complained about
>that too.
```

Because newbies know better.

From: **Jan Wielemaker**
Subject:Advice from a newbie

anyway <b@rt.luna.be> wrote:
```
> Morin Thurgh wrote:
>
>> the database alteration by means of retract and assert*
is not
>> intended to be particularly efficient
> [...]
>> This is to plainly say that your code looksccorrect, but
neither
>> particularly innovative, nor actually efficient.
>>
>> The lack of efficiency in altering the knowledge base
>
> I am glad you raise the efficiency point - which I also
did, but I was
> talking about complexity rather. Isn't efficiency a
quality of the
> implementation of a language, not (necessarily) of the
language
> itself. Or do you imply that assert* and retract must be
necessarily
> inefficient in Prolog ?
```

Well, by definition assert *copies* what you assert and (as was pointed out) therefore you end with list holding copies of the original list. The traditional reverse/2 implementation does not copy the content of the list *elements*. That is not only fundamentally faster, but it also fundamentally different. Otherwise copy_term/2 would not exist :-)

Or do you wish to argue that after my_copy_term below it is allowed that Copy == Term if Term is not ground?

```
my_copy_term(Term, Copy) :-
    assserta(tmp(Term)),
    once(retract(tmp(RawCopy))),
    RawCopy = Copy.
```

Cheers --- Jan

P.s. Wondering how many real newbies are not already asleep after concluding that Prolog programmers are completely nuts ...

EDITORIAL INFORMATION

VOL. 21 NO. 1
FEBRUARY/MARCH 2008

The ALP newsletter

What is the ALP Newsletter?

This is the electronic newsletter of the Association for Logic Programming (ALP, http://www.logicprogramming.org/). It contains news, net postings, call for papers, comment, conference announcements and humour, all related to Computational Logic.

The newsletter is a quarterly publication, in the months February, May, August and November a new issue is posted.

To remind interested people of the outcome of a new issue, a short digest is sent by email to those who subscribe to it.

The digest is a service anyone can subscribe to, either via web at http://listserv.surfnet.nl/archives/alp.html, or via email (see instructions below).

We guarantee that subscribers won't receive from us more emails that strictly necessary - four or five emails PER YEAR is all we are going to send around. It goes without saying that subscribing is free, and that the email addresses in the list will **never** - under any circumstance - be given to any third party.

Subscribe/Unsubscribe

How to subscribe to the digest.

a) via web-interface at http://listserv.surfnet.nl/archives/alp.html. (please tick **regular** in the subscription type)

b) by sending an email to LISTSERV@NIC.SURFNET.NL with in the **BODY ONLY** the line "subscribe alp **FIRSTNAME LASTNAME**", where **FIRSTNAME** and **LASTNAME** are - of course - your first and last name. If you prefer to remain anonymous, send the line "subscribe alp anonymous" instead.

How to unsubscribe to the digest: there are two simple ways.

a) via web-interface at http://listserv.surfnet.nl/archives/alp.html

b) by sending Just send an email LISTSERV@NIC.SURFNET.NL with in the **BODY ONLY** the line "**SIGNOFF ALP**".

Newsletter Submissions are Welcome!

Please send us anything you think will be of interest to newsletter readers. For instance:

- news on Computational Logic related products and services;
- letters and comments;
- abstracts or reviews of papers and books related to logic programming;
- short articles of general interest (1-2 pages);
- your views on any aspect of LP;
- conference reports;
- calls for papers and announcements of LP related workshops and conferences;
- puzzles and humorous notes, etc.;
- suggestions for articles and themes for future editions.

If you have an idea and are unsure about its suitability, do email me or one of the area editors to discuss it further.

Submissions have to be either in plain text or html. Latex submissions are also accepted, as they can be transformed in html via LaTeX2html. No other formats are accepted.

Enrico Pontelli, epontell aT cs.nmsu.edu

Who is who

- **Newsletter Editor: Enrico Pontelli**
  - E. Pontelli: New Mexico State University - USA. http://www.cs.nmsu.edu/~epontell/.
- **Area Editors:**
  - **Implementation & NetTalk: Roberto Bagnara**
    University of Parma, Italy
    http://www.cs.unipr.it/~bagnara/
  - **Games, Puzzles, and Applications: Paolo Baldan**
    University Ca' Foscari, Venice, Italy
    http://www.dsi.unive.it/~baldan
  - **Theorem Proving: Brigitte Pientka**
    McGill University, Canada
    http://www.cs.mcgill.ca/~bpientka/
  - **Constraints: Eric Monfroy**
    University of Nantes, France
    http://www.sciences.univ-nantes.fr/info/perso/permanents/monfroy/
  - **Concurrency: Frank Valencia**
    Lix, Ecole Polytechnique de Paris, France
    http://www.brics.dk/~fvalenci
  - **Verification and Model Checking: C.R. Ramakrishnan**
    SUNY Stony Brook, USA
    http://www.cs.sunysb.edu/~cram/
  - **Multi-Agent Systems: Fariba Sadri and Francesca Toni**
    Imperial College, London, UK
    http://www-lp.doc.ic.ac.uk/UserPages/staff/fs/
    http://www-lp.doc.ic.ac.uk/UserPages/staff/ft/ft.html
  - **Non-Monotonic Reasoning: Tran Cao Son**
    New Mexico State University, USA
    http://www.cs.nmsu.edu/~tson
  - **Applications of Logic and Constraint Programming: Agostino Dovier**
    University of Udine, Italy
    http://www.dimi.uniud.it/~dovier
  - **Web and Semantic Web: Axel Polleres**
    Universidad Rey Juan Carlos, Spain
    http://www.polleres.net
- **Webmaster: Andrew O. Gonzalez**