Association for Logic Programming Newsletter

Vol. 19, No. 4
November/December 2006

The ALP newsletter

What is the ALP Newsletter?

This is the electronic newsletter of the Association for Logic Programming (ALP, http://www.cwi.nl/projects/alp). 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**".

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**

Contents

Vol. 19 n. 4, November/December 2006

Editorial

Enrico Pontelli

Dear Logic Programmers,

Welcome to the November 2006 issue of the ALP Newsletter.

First of all, my apologies for the delays in the publications of this issue - it is almost one month late! As you can see, we experimented with a new design and things initially did not go as smooth as expected. But I hope you like the result. We have a new Web developer on board - Andrew Gonzalez - who will be helping us with the addition of new features to the newsletter. You will see many additions in the coming issues! Please, do not hesitate to send me any suggestions/comments/ critics/praises/... you may have, including things you would like to see (or things you don't want to see...). Remember, this is your newsletter. Some of the projects we are currently implementing is the development of a "LP Map".

And since we are talking about this, please let me re-iterate my invitation for submissions. If you have any piece of information that you would like to share with the LP community (call for papers, software announcements, some startling discoveries, a graduating student...) please send them to me. The newsletter can only grow with the input of all the members of the community (and there are many of you out there). In particular, I have always been a strong advocate of students and student participation, so please help me in improving visibility of the good work our students are performing; it will help them, it will help attracting more students to our community, and it will demonstrate to the world that LP is alive and kicking. In particular, I am still desperately looking for submissions to the Doctoral Dissertations column - if you have even MS students finishing some interesting MS Project/Theses in the field of LP, I would be welcome to include those as well. And we will have another Doctoral Consortium in 2007 in Porto, please encourage your students to apply, it is a fantastic experience (whoever attended the Consortium in 2006 can confirm this...) and it provides students with some funds to attend ICLP (we assembled a pretty good financial packet in 2006).

We had a fantastic time at ICLP 2006. Seattle welcomed us with an unusually sunny and warm weather, and the organization of FLoC provided us with a very good event. ICLP was very pleasant, the talks were good (I particularly enjoyed the invited talks) and the range of workshops was outstanding - it was actually

hard to choose which workshop to attend, as many of them provided very interesting material. I found particularly interesting the Open Session provided in the Semantic Web workshop (you can read about it in the Workshop Report in this issue of the newsletter), a very interesting approach which was enthusiastically welcomed by the audience. Overall, ICLP 2006 was a success, so please join me in thanking Sandro, Mirek, Manuel, and all the organizers for their hard work and commitment!

And as one ICLP passes by, another one moves our way... ICLP 2007 will be in Porto, and Fernando Silva is planning great things for us - if you are familiar with the traditional Portuguese hospitality, you know that it can only mean an event to really look forward to. Please keep an eye on the various announcements and call for papers that have already started circulating (you can find the preliminary CFP in this issue as well).

ALP is also soliciting formal proposals for ICLP 2008, so if you feel up to the challenge please contact Gopal Gupta.

I need to close this with a very sad news. It is always terrible to announce that a member of our community has departed; Marco Cadoli, a well-established researcher in LP (and other areas) has lost his battle against an incurable disease. In this issue you will find some words in his memory by Marco Schaerf and Jack Minker. The ALP Newsletter team would like to extend condolences to Marco's family and friends.
To bring this one to closure, I would like once again to welcome your comments/ critics/suggestions/... on how you would like to see the ALP Newsletter evolve in the near and not-so-near future.

Happy Holidays!

Enrico

In Memoriam
Marco Cadoli (1965-2006)

Marco Schaerf
University of Rome "La Sapienza"
Italy

Editor: Agostino Dovier

Dear friends,

With deep sorrow I inform you that on November 21st 2006 our friend and colleague Marco Cadoli passed away at the eraly age of 40. Marco has fought a rare form of cancer for three years always showing an incredible strength, continuing his work research work till the very end.

Marco was a researcher deep at heart, who was always thrilled to find new ideas, new applications and always eager to discuss with colleagues, students and friends. His research has spanned through many areas such as Artificial Intelligence, Databases, Logic Programming, Constraint Programming and formal methods in Software Engineering.

Soon after his death the mailboxes of the closest colleagues, such as Maurizio Lenzerini and myself, have been filled with condolences messages. I want to include here the message Maurizio received from Jack Minker.

Marco Schaerf

> >----------Message by Maurizio Lenzerini posted on Dbworld----
> >With deep sorrow I inform the scientific community that Marco Cadoli
> >(**http://www.dis.uniroma1.it/~cadoli/**) has passed away on
> November 21,
> >2006, after a long battle with cancer. He leaves his beloved wife
> >Laura and little sons Andrea and Riccardo.
> >

>Marco was a brilliant researcher, a wonderful person, and an
>invaluable friend. The DB&KR group at University of Rome ``La
>Sapienza'' will be forever proud to have had him as one of its
>member.
>
>Maurizio
>
>--------personal reply by Jack Minker----------
>Dear Maurizio,
>
>I was deeply saddened to learn that Marco died yesterday. I looked
at
>his web page and again saw this young, intelligent and handsome
man
>whose work I admired. I looked over his vitae and was again
impressed
>by the research he contributed to the database and logic-based
>computing community. I also found that he was a deeply religious
>person which, I trust, was a comfort to him and his family.
>
>As you say, he was, indeed a brilliant researcher, a wonderful
person
>and an invaluable friend. I hope that the logic programming
community
>at its next international meeting will, at the least, have a moment of
>silence in memory of Marco.
>
>Please give my condolences to Marco's family and to your
colleagues
>who had the pleasure of working with this wonderful person. My
>condolences also to you with whom he wrote outstanding papers
and with whom
>you were a friend.
>It is a deep loss to the entire DB and LP scientific community to lose
>this decent person and outstanding researcher.
>
>Sadly,
>
>Jack

Timed Concurrent Constraint Programming in Systems Biology

Alejandro Arbeláez, Julian Gutiérrez and Jorge A. Pérez
AVISPA Research Group*, Department of Science and Engineering of Computing
Pontificia Universidad Javeriana, Cali, Colombia
{aarbelaez,jg,japerez}@cic.puj.edu.co

3rd November 2006

Abstract

Systems biology aims at getting a higher-level understanding of living matter, building on the available data at the molecular level. In this field, theories and methods from computer science have proven very useful, mainly for system modeling and simulation. Here we argue that languages based on *timed* concurrent constraint programming (timed ccp) —a well-established model for concurrency based on the idea of partial information— have a place in systems biology. We summarize some works in which our group has tried to assess the possibilities/limitations of one such formalisms in this domain. Our base language is `ntcc`, a non-deterministic, timed ccp process calculus that provides a *unified framework* for modeling, simulating and *verifying* several kinds of biological systems. We discuss how the interplay of the operational and logic perspectives that `ntcc` integrates greatly favors biological systems analysis.

1 Introduction

Recent years have seen an extraordinary progress in the field of molecular biology. The enormous amount of biological data gathered in the last years has generated a paradigm shift, in which giving such data a coherent meaning is now a growing necessity for researchers. The interest is then to identify and understand *biological functions* building on the available knowledge on *basic elements* such as proteins and genes. This requires following a *system-level approach* where isolated data is structured as to make up interactions that, in turn, will constitute more complex interactions at a higher level of abstraction. This is, broadly speaking, the goal and motivations of what it is commonly referred to as *systems biology*.

Process calculi are abstract specification languages in which the notions of *process* and *interaction* prevail in the formalization of systems exhibiting concurrent behavior. There is a natural correspondence between the kinds of interactions provided by process calculi and those present in biological systems. The simplicity of such correspondence has captured the attention of experts in both domains. As a matter of fact, calculi for mobile processes have been used in the biological context (see, e.g., [24]), and new calculi focusing on particular aspects of biological interactions have been proposed (see [19] for a survey). This research direction is sometimes called the *language approach* for systems biology. Briefly, the idea consists in defining some *working analogies* between both biological entities and phenomena and the elements of the calculus. Evolution of biological systems can be then formalized by means of some (operational) semantics provided by the calculus. In this context, most process calculi only offer modeling and simulation capabilities.

Although this language approach has shed light on the nature of several biological systems, we believe that logic-based reasoning techniques could effectively complement biological systems analysis. More precisely, we argue for process calculi based on *timed* concurrent constraint programming (ccp) [27, 26] for analyzing biological processes. Since process terms in ccp can be viewed at the same time as computing

*`http://avispa.puj.edu.co`

agents and logic formulas, it can constitute a unified framework where biological systems can be described, simulated and also *verified*. This paper aims at supporting this claim by summarizing our initial efforts in this direction. Our base language is ntcc [16], a non-deterministic, discrete time process calculus based on ccp. Below we elaborate on how two salient features of ntcc—the interplay of partial information and non-determinism and its logic-based reasoning techniques— can be convenient in the biological context.

Partial information arises naturally in the description of biological systems. Biologists usually count with partial information obtained from experimental measurements over the systems of interest; such information should be exploited as much as possible so that working hypothesis can be refined and, at the end, new experiments can be better oriented [15]. In the spirit of the language approach, and from a more abstract level, partial information can be classified as *quantitative* and *behavioral*. While *partial quantitative information* usually involves incomplete information on the *state of the system* (e.g., the set of possible values that a variable can take), *partial behavioral information* refers to the uncertainty in the behavior of interactions (e.g., the unknown relative speeds on which two systems interact).

These two kinds of partial information are naturally captured in ntcc. On the one hand, partial quantitative information is captured by the notion of *constraint system*, a structure that defines logic inference capabilities over constraints. Constraint systems are *parametric* to ntcc, which allows to state several kinds of conditions by choosing the appropriate constraint system(s). On the other hand, partial behavioral information is represented by *non-deterministic* and *asynchronous operators* available in ntcc. We shall see how the interplay of these operators in the discrete time of ntcc allows to explicitly describe and reason about the uncertainty in the time occurrence of many biological phenomena.

Reasoning techniques in ntcc allow to prove whether a given process P satisfy a given property F, using a linear-temporal specification logic and its corresponding proof system. The symbolic flavor conveyed by logic-based verification can effectively complement conventional simulations when analyzing biological systems involving partial behavioral information. In fact, conventional simulations of biological components which, e.g., act at unknown or unpredictable times, might not faithfully reflect the possible behavior of the system. This kind of inaccuracies could be observed in simulations independently of how powerful simulation tools are. This is but one situation where counting with logic-based reasoning tools would come in handy for complementing analysis of biological systems.

We shall take advantage of these features by modeling biological systems as processes and their properties as linear-temporal formulas, all *in a single framework* in which non-determinism and partial information are essential. An additional advantage of using ntcc for the study of biological systems consists in the possibilities of turning this theoretical framework into software tools. As a matter of fact, our group has built ntccSim [4, 2], a simulation tool that admits the description of biological systems as ntcc processes and allows to observe their behavior over time. Formalisms, methods and tools from timed ccp therefore constitute a real alternative for biological systems analysis.

Plan of the document Section 2 further discusses the intuitions underlying systems biology outlined above. Section 3 introduces the ntcc calculus in a biological context. Section 4 discusses some biological systems that have been analyzed with our approach. Some related work is reviewed in Section 5. Section 6 gives some concluding remarks and proposes directions for future work.

2 Systems Biology

Recent progresses in *molecular biology* have allowed to describe the structure of many components making up biological systems (e.g., genes and proteins) as *isolated* entities. Instead of being alone, these entities are part of complex biological networks present at the cellular environment (such as, e.g., genetic regulatory networks) which define and regulate cellular processes. The current challenge is to move from molecular biology to *systems biology* [14, 15], in order to understand how these individual components and entities *integrate* to each other in the networks they shape. Once this integration has been understood, it will be then possible to discover how these entities perform their tasks.

Systems biology then aims at studying the mechanisms by which genes and proteins integrate and interact among them inside an organism. That is, systems biology studies in an integrated way both the structure and expression of a gene or a set of genes. The notions of *system* and *multilevel interaction* are crucial in this

study. The former is justified by the need of considering the interactions within the given system, under the assumption that components of a system are not isolated and therefore influence each other. The latter refers to the capability of analyzing the same biological system, observing and abstracting its essential properties, at different levels of detail.

The complexity and size of biological systems motivates the use of computational techniques that allow to build models of these systems that *abstract* their behavior and make their study easier. More precisely, in a hypothesis-driven research approach [15], computing techniques are at the start of a four-stage research cycle that is complemented by analysis, technology and genomics phases. The idea is to use computer-based techniques to simulate biological models representing contradictory issues of biological significance. These so-called *dry experiments* should reveal inadequacies of the assumptions embedded in models and, after a phase where simulation results are analyzed and theories formulated, they are the basis for *wet (real) experiments*. Finally, the successful experiments will be those that eliminate inadequate models.

In this context, where the interest is to *refine* models by progressive simulations, existing languages and formalisms from concurrency theory can be convenient. Notice that the above-described hypothesis-driven approach heavily depends on the appropriate use of partial information in simulations. Moreover, counting with hypothesis suggest that the use of logic methods for their analysis is reasonable. We now enter to describe a suitable framework for carrying out this kind of analysis.

3 Timed Concurrent Constraint Programming

Here we give a concise, informal introduction to `ntcc`, the process calculus that we have used to model and verify biological systems. Based on [12], we focus on how `ntcc` constructs can be convenient in the biological context. The interested reader is referred to [16] for an in-depth presentation of `ntcc`.

We start by briefly discussing some basic notions of concurrent constraint programming (ccp), a well-established formalism for concurrency which generalizes Logic Programming [25]. One of the most appealing and distinctive features of ccp is that it combines the traditional *operational* view of process calculi with a *declarative* one of processes based upon logic. This combination allows ccp to benefit from the large body of techniques of both process calculi and logic.

In ccp the knowledge about the system is expressed in terms of *constraints*, or statements defining the possible values a variable can take (e.g., $x + y \geq 7$). These pieces of partial information are *monotonically* accumulated in shared medium, so-called *store*. Processes (or agents) then interact with each other by *telling* and *asking* constraints to the store. They synchronize according to the information in the store.

One fundamental notion in ccp is that of a *constraint system*. Informally, a constraint system provides a signature from which constraints can be constructed, and an entailment relation which specifies the inter-dependencies among them. For operational reasons, we shall require this relation to be decidable. A practical example of a constraint system is FD [13]. In FD variables are assumed to range over finite domains and, in addition to equality, we may have predicates that restrict the possible values of a variable to some finite set.

3.1 The `ntcc` process calculus

The `ntcc` process calculus [16] is a *temporal* extension of ccp. Its process constructs naturally capture the main features of timed and reactive systems. In particular, `ntcc` allows to model:

- *non-determinism* to express diverse execution alternatives for a system from the same initial conditions.
- *asynchrony* to represent unbounded but finite delays in the execution of a system.
- *unit-delays* to explicitly model pauses in system execution.
- *time-outs* to express the possibility of default behavior, reasoning about the absence of information.
- *synchrony* to control and coordinate the concurrent execution of multiple systems.
- *infinite behavior* to represent the persistent execution of a system.

ntcc formalizes discrete, reactive computation. In ntcc, time is conceptually divided into *discrete intervals* (or *time units*). In a particular time unit, a process P gets an input c from the environment (an item of information represented as a constraint), it executes with this input as the initial store, and when it reaches its resting point, it outputs the resulting store d to the environment. The resting point determines a residual process P', which is then executed in the next time unit. Notice that information is not automatically transferred from one time unit to the following.

Process Syntax

In ntcc, processes $P, Q, \ldots \in Proc$ are built from constraints $c \in \mathcal{C}$ and variables $x \in \mathcal{V}$ in the underlying constraint system by:

$$\begin{array}{rcllll} P, Q, \ldots & ::= & \mathbf{tell}(c) & \mid \sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i & \mid P \parallel Q & \mid \mathbf{local}\ x\ \mathbf{in}\ P \\ & \mid & \mathbf{next}\ (P) & \mid \mathbf{unless}\ c\ \mathbf{next}\ P & \mid \star\ P & \mid !\, P \end{array}$$

Below we provide some intuitions regarding the behavior of ntcc processes.

Including and Querying (Partial) Information Process $\mathbf{tell}(c)$, the simplest operation to express *partial information*, includes a constraint c into the current store, thus making it available to other processes in the same time interval.

In the biological context, $\mathbf{tell}$ operations allow to represent at least two kinds of *partial information* statements: so-called *ground rules* and *state definition* statements. The first ones precisely state certain conditions that apply during the life of the biological system. These conditions can easily exploit the available (possibly incomplete) knowledge. Complementary, *state definition* statements refer to those constraints intended to define the exact values for the variables in the system. This is particularly useful when one exactly knows the set of possible states for the system at a given time; series of such statements (for different time units) thus constitute a detailed view of the behavior of the system. Remarkably, the *declarative flavor* in both kinds of statements could favor the definition of essential properties in (biological) models.

Guarded operations of the form $\mathbf{when}\ c\ \mathbf{do}\ P$ complement $\mathbf{tell}$ operations and constitute the basic means for *querying* (or *asking*) information about the state of a system. Intuitively, a $\mathbf{when}\ c\ \mathbf{do}\ P$ process queries the current constraint store: if the guard c is present in such a store then the execution of P is enabled. The "presence" of c depends on the inference capabilities associated with the store. That is, a particular constraint could not be explicitly present in the store, but it could be inferred from the available information. It is straightforward to interpret $\mathbf{when}$ operations as a way of expressing the *preconditions* for reaching a particular state of the system. The behavior of the system can be precisely stated in this way.

Non-deterministic Choices Non-determinism allows to represent several possible courses of action from the same initial state, without providing any information on how one of such courses is selected. In ntcc, non-deterministic behavior is obtained by generalizing processes of the form $\mathbf{when}\ c\ \mathbf{do}\ P$: a *guarded-choice summation* $\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$, where I is a finite set of indexes, represents a process that, in the current time interval, must non-deterministically choose one of the P_j $(j \in I)$ whose corresponding constraint c_j is entailed by the store. The chosen alternative, if any, precludes the others. If no choice is possible then the summation is precluded. We use $\sum_{i \in I} P_i$ as an abbreviation for the "blind-choice" process $\sum_{i \in I} \mathbf{when}\ \mathtt{true}\ \mathbf{do}\ P_i$. We use $\mathbf{skip}$ as an abbreviation of the empty summation and "$+$" for binary summations.

In the biological context, the combination of guarded choices and partial information represents an appropriate mechanism to formalize the inherent *unpredictability* in system interactions. In this sense, non-deterministic choices allows to explicitly represent *partial behavioral information*.

Communication Process $P \parallel Q$ represents the parallel composition of P and Q. In one time unit P and Q operate concurrently, "communicating" via the common store by adding and querying information. We use $\prod_{i \in I} P_i$, where I is a finite set of indexes, to denote the parallel composition of all P_i.

Local Information In ntcc, processes of the form $\mathbf{local}\ x\ \mathbf{in}\ P$ behave like P, except that all the information on x produced by P can only be seen by P and the information on x produced by other processes cannot be seen by P.

Although the conventional spirit of this kind of operators is to restrict the interface through which a process can interact with each other, in the context of partial information local information may represent a valuable help in the analysis of systems. When performing overall analyzes of complex systems, local variables may help to "hide" the behavior of those components that are irrelevant in the interactions to be analyzed. The interplay of hiding and partial information may allow to analyze systems at different levels of detail.

Basic Timed Behavior The basic time operator in ntcc is $\mathbf{next}\ (P)$, which represents the activation of P in the next time interval. That is, $\mathbf{next}\ (P)$ models a *unit-delay* of process P. It can be also considered as the simplest way of expressing dynamic behavior over time. This is fundamental in ntcc, since information is *not automatically* transferred from one time interval to the next. Based on $\mathbf{next}\ (P)$, more sophisticated delay constructs can be defined: we use $\mathbf{next}^n\ (P)$ as an abbreviation for $\mathbf{next}\ (\mathbf{next}\ (\ldots \mathbf{next}\ (P))\ldots))$, where $\mathbf{next}$ is repeated n times.

Absence of Information / Unexpected Behavior In the biological setting, to be able of reasoning about *absence* of information is both important and necessary. Although sometimes it is possible to predict some of the possible future states for a system, usually there is a strong need of expressing *unexpected behavior*. In this kind of scenarios, processes of the form $\mathbf{unless}\ c\ \mathbf{next}\ P$ may come in handy: P will be activated only if c cannot be inferred from the current store. The "unless" processes thus add (weak) time-outs to the calculus, i.e., they wait one time unit for a piece of information c to be present and if it is not, they trigger activity in the next time interval.

Asynchrony The $\star$ operator allows to express asynchronous behavior through the time intervals. Process $\star P$ represents an arbitrary long but finite delay for the activation of P.

This kind of asynchronous behavior therefore constitutes another instance of partial behavioral information: in addition to the partial information *on the variables* that are part of the state of the system (and that can be expressed by the operators discussed above), the $\star$ operator allows to express partial information *on the time units* where processes are executed. This is particularly interesting when describing (biological) processes that interact at *unknown relative speeds*.

The partial information spirit of the asynchronous behavior in ntcc is strengthened by the following derived operator, expressing *bounded eventuality*:

$$\star_{[n,m]}\ P = \mathbf{next}^n\ (P) + \mathbf{next}^{n+1}\ (P) + \cdots + \mathbf{next}^{m-1}\ (P) + \mathbf{next}^m\ (P).$$

This temporal operator thus represents an additional amount of partial information, as it ensures that P will be activated at some point within the time units in the closed interval of naturals $[n, m]$. As in the original operator, there is no additional information of when this restricted eventuality will take place.

Persistent Behavior Somehow opposed to the eventual behavior enforced by asynchronous behavior, *persistent* (or infinite) behavior serves to express conditions that are valid during every possible state of the system. The *replication* operator $!P$ represents $P \parallel \mathbf{next}\ (P) \parallel \mathbf{next}^2(P) \parallel \ldots$, i.e. unboundedly many copies of P but one at a time. As such, persistent behavior is an appropriate way of enforcing conditions stating ground rules of the systems of interest. It also can also be understood as a mechanism that allows to move from *static* descriptions or conditions (valid only in one state of the system) to *dynamic* statements that are always valid.

As in the asynchronous case, it is possible to derive a bounded version of the persistent operator:

$$!_{[n,m]}\ P = \mathbf{next}^n\ (P) \parallel \mathbf{next}^{n+1}\ (P) \parallel \cdots \parallel \mathbf{next}^{m-1}\ (P) \parallel \mathbf{next}^m\ (P).$$

This operator represents the fact that P is always active during all the time units in the interval $[n, m]$. As its eventual counterpart, this derived operator (known as *bounded invariance*) may come in handy when certain additional information regarding the (persistent) execution of P is available.

LTELL	$\mathbf{tell}(c) \vdash c$	LSUM	$\dfrac{\forall i \in I \quad P_i \vdash A_i}{\sum_{i\in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i \vdash \dot{\bigvee}_{i\in I}(c_i \mathbin{\dot{\wedge}} A_i) \mathbin{\dot{\vee}} \dot{\bigwedge}_{i\in I} \dot{\neg} c_i}$			
LPAR	$\dfrac{P \vdash A \quad Q \vdash B}{P \parallel Q \vdash A \mathbin{\dot{\wedge}} B}$	LUNL	$\dfrac{P \vdash A}{\mathbf{unless}\ c\ \mathbf{next}\ \ P \vdash c \mathbin{\dot{\vee}} \bigcirc A}$			
LREP	$\dfrac{P \vdash A}{!P \vdash \Box A}$	LLOC	$\dfrac{P \vdash A}{\mathbf{local}\ x\ \mathbf{in}\ P \vdash \dot{\exists}_x A}$			
LSTAR	$\dfrac{P \vdash A}{\star P \vdash \Diamond A}$	LNEXT	$\dfrac{P \vdash A}{\mathbf{next}\ (P) \vdash \bigcirc A}$	LCONS	$\dfrac{P \vdash A}{P \vdash B}$	if $A \Rightarrow B$

Table 1: A proof system for (linear-temporal) properties of ntcc processes

A Logic Approach for Property Verification

ntcc is associated with a linear-temporal logic, which is defined as follows. Formulas $A, B, \ldots \in \mathcal{A}$ are defined by the grammar:

$$A, B, \ldots := c \mid A \dot{\Rightarrow} A \mid \dot{\neg} A \mid \dot{\exists}_x A \mid \bigcirc A \mid \Box A \mid \Diamond A.$$

Here c denotes an arbitrary constraint which acts as an atomic proposition. Symbols $\dot{\Rightarrow}$, $\dot{\neg}$ and $\dot{\exists}_x$ represent linear-temporal logic implication, negation and existential quantification. These symbols are not to be confused with the logic symbols $\Rightarrow$, $\neg$ and $\exists_x$ of the constraint system. Symbols $\bigcirc$, $\Box$ and $\Diamond$ denote the linear-temporal operators *next*, *always* and *eventually*. We use $A \mathbin{\dot{\vee}} B$ as an abbreviation of $\dot{\neg} A \dot{\Rightarrow} B$ and $A \mathbin{\dot{\wedge}} B$ as an abbreviation of $\dot{\neg}(\dot{\neg} A \mathbin{\dot{\vee}} \dot{\neg} B)$. The standard interpretation structures of linear temporal logic are infinite sequences of states. In ntcc, states are represented with constraints, thus we consider as interpretations the elements of $\mathcal{C}^\omega$. When $\alpha \in \mathcal{C}^\omega$ is a model of A, we write $\alpha \models A$.

We shall say that P satisfies A if every infinite sequence that P can possibly output satisfies the property expressed by A. A relatively complete proof system for assertions $P \vdash A$, whose intended meaning is that P satisfies A, is given in Table 1. We shall write $P \vdash A$ if there is a derivation of $P \vdash A$ in this system.

4 Using ntcc for Analyzing Biological Systems

In this section we describe how we have applied our approach for biological systems analysis in several kinds of systems. These include: mechanisms for active transport of substances through cellular membranes, genetic regulatory networks (GRNs), and mutations over a GRN. We briefly comment on the nature of the modeled systems and describe their ntcc models.

4.1 Active Transport in Cellular Membranes

In [12] we have used ntcc to model and verify an *ion pump*, a natural channel connecting the two sides of a membrane. These pumps move ions across the membrane in a process called *transport*. Depending on the source of the required energy, the transport can be either *passive* or *active*. In passive transport ions freely move across the membrane following an electrochemical gradient, so the cell does not need to provide energy for the transport. Since in active transport ions move against the direction of the gradient, the cell has to supply energy (usually in form of ATP) to accomplish this movement.

The Sodium-Potassium pump [28] (SP-pump in the sequel) is a system for active transport in animal cells. It exchanges Sodium ions inside the cell with Potassium ions outside of it. The pump is composed of two proteins known as the alpha and beta subunits. The purpose of the pump is to keep the concentration of sodium inside the cell lower than outside. This difference of concentrations generates an electrochemical gradient that leads the passive transport of Sodium ions towards the cytoplasm in the cell. If the pump does not work well then the gradient becomes weak for transport, thus affecting the entrance of required substances into the cell.

The pumping process in the SP-pump can be divided in six phases. At the beginning there is a pump conformation with high affinity for Sodium ions inside the cell (1). This conformation encourages the binding of three Sodium ions with the pump. Then the alpha subunit is phosphorylated by ATP hydrolysis (2), leaving a residual ADP molecule in the cytoplasm. This chemical reaction provides the needed energy for the pumping process. Once this occurs, the pump conformation changes and then the Sodium ions can leave the cell (3). At this point, there is a pump conformation with high affinity for Potassium ions outside the cell (4). This results in the binding of two Potassium ions with the pump. Hence, the alpha subunit is dephosphorylated (5) and the pump conformation returns to the initial state. At this moment Potassium ions can enter the cell (6). The pumping process is always performed regulating the concentration of Sodium in the cell.

In parallel to this active transport movement, there is a *passive* transport movement that allows Potassium and Sodium ions to move against the direction of the active transport. This complementary movement is induced by an electrochemical gradient present in the cell.

Elements of an `ntcc` model of the SP-pump

Here we describe the main principles underlying the `ntcc` model of the SP-pump. We use non-deterministic and asynchronous behavior for modeling partial behavioral information regarding temporal responses of certain components. We use mutable entities (*cells*) and recursive definitions in some of our models. Cells can be easily encoded in `ntcc`; see [16] for more details.

The model assumes a constraint system over finite domains of integers, considering three places for interaction: inside and outside the cell, and an intermediate place where ions stay before entering or flowing out of the cell (i.e., the pump). The model involves a series of cells that store useful quantities about the pumping process. We use notations $x : v$ and $x := v$ to represent the *initialization* and the *assignment* of a cell x with value v, respectively. Output and input operations of the pump are then modeled as modifications over variables representing the number of ions both inside and outside the cell. In this way, for instance, variables Na_O and Na_I represent the amount of Sodium ions placed outside and inside the cell, respectively. In addition, a certain amount of each kind of ion needed for the correct functioning of the cell is assumed. Such amounts are denoted by Na_{IDEAL} and K_{IDEAL}. Moreover, some additional variables capture other details of the pump: $OPump$ represents the orientation of the pump (either inside or outside the cell), $Alpha$ denotes the current binding of the alpha subunit and $Pump$ represents the current content of the pump. These three variables will be instantiated with constants that can be encoded by integers: for instance, possible values for $Alpha$ are `P`, `free` and `null` (note the special font style given to constants). Finally, integer variables ATP and ADP represent the presence of ATP and ADP inside the cell, respectively.

The complete model for the SP-pump (denoted as the $NaKPump$ process) reflects the complementary nature of active and passive transport in the SP-pump, and is represented by the integration of $ActiveTrans$ and $PassiveTrans$ processes. From this $NaKPump$ process it is then possible to assume some environment in which the pump is placed. This is the intuition behind process $System$. We now proceed to explain in a greater detail the ideas behind these processes. For the sake of space, we only include fragments of the model; the interested reader is referred to [12] for complete details.

Active Transport Phases Process $ActiveTrans$ integrates sub-processes for the six phases described before; these processes invoke each other. Some processes include recursive calls to themselves. This intends to represent the possibility that the system remains stuck in certain phases, even if all the conditions needed to evolve are given. That is, we are trying to model "reversible" phases, a behavior that is represented by non-deterministic choices. As a result, those phases could be executed several times therefore delaying system execution in at least one time unit. Such a delay occurs because the system waits for the presence of some substances at a specific place of the pump. In fact, those substances could be available but not in the required place. Figure 1 presents a fragment of $ActiveTrans$ in which the phases where the Sodium leaves out the cell are represented. Process $NaPhase2$ is the only reversible phase.

The above-described non-deterministic and asynchronous behavior could represent other conditions on component binding, such as an appropriate physical contact among elements that (chemically) react with components of the pump. Similarly, non-deterministic behavior can also represent some kind of malfunction.

$$
\begin{aligned}
NaPhase1 &\stackrel{\text{def}}{=} \mathbf{when}\,(Na_I > Na_{IDEAL} \vee K_I < K_{IDEAL}) \wedge Pump = \texttt{Empty} \wedge OPump = \texttt{In}\ \mathbf{do}\\
&\quad (\mathbf{next}\,(Na_I := Na_I - 3 \parallel Pump := \texttt{Na} \parallel \mathbf{tell}(unchangedK = 1) \parallel NaPhase2)\ +\\
&\qquad \mathbf{next}\,(NaPhase1 \parallel \mathbf{tell}(unchangedK = 1) \parallel \mathbf{tell}(unchangedNa = 1)))\\
NaPhase2 &\stackrel{\text{def}}{=} \mathbf{when}\, Pump = \texttt{Na} \wedge Alpha = \texttt{free} \wedge ATP > 0\ \mathbf{do}\\
&\quad (\mathbf{next}\,(OPump := \texttt{Out} \parallel Alpha := \texttt{P} \parallel ADP := 1 \parallel\\
&\qquad \mathbf{tell}(unchangedK = 1) \parallel \mathbf{tell}(unchangedNa = 1) \parallel NaPhase3)\\
&\qquad + \mathbf{next}\,(NaPhase2 \parallel \mathbf{tell}(unchangedK = 1) \parallel \mathbf{tell}(unchangedNa = 1)))\\
NaPhase3 &\stackrel{\text{def}}{=} \mathbf{when}\, Pump = \texttt{Na} \wedge OPump = \texttt{Out}\ \mathbf{do}\\
&\quad \mathbf{next}\,(Na_O := Na_O + 3 \parallel Pump := \texttt{Empty} \parallel \mathbf{tell}(unchangedK = 1) \parallel KPhase1)
\end{aligned}
$$

Figure 1: Fragment of the ntcc model for the active transport phases of the Sodium-Potassium pump

$$
\begin{aligned}
PassiveNa &\stackrel{\text{def}}{=} \mathbf{unless}\ Na_O = Na_I\ \mathbf{next}\\
&\quad (\mathbf{next}^5\,(PassiveNa) \parallel\\
&\qquad \star_{[0,5]}(\mathbf{unless}\ unchangedNa = 1\ \mathbf{next}\ \ (Na_I := Na_I + 3 \parallel Na_O := Na_O - 3) \parallel\\
&\qquad\quad \mathbf{when}\, unchangedNa = 1\ \mathbf{do}\ \ (Na_I := Na_I + 3 \parallel Na_O := Na_O - 3)))\\
PassiveTrans &\stackrel{\text{def}}{=} PassiveNa \parallel PassiveK
\end{aligned}
$$

Figure 2: Fragment of the ntcc model for the passive transport phases of the Sodium-Potassium pump

For instance, in phase $NaPhase2$ the phosphate could not bind to the alpha subunit, which would result in a malfunction of the system that could be directly observed from the evolution of the pump in time.

Passive Transport Phases Passive transport is represented by process $PassiveTrans$, which defines two sub-processes: one for the entrance of Sodium ions and another for the output of Potassium ions. In the modeling of these sub-processes we consider partial behavioral information on the actual time when the ion movement really occurs, which is represented by a bounded asynchronous operator. Figure 2 describes a fragment of $PassiveTrans$.

Additional Processes The integration of the above processes as in the $NakPump$ process is straightforward. There is an additional process (i.e., $Control$) which governs the global behavior of the pump w.r.t. the equilibrium of the ions amounts; in the case an equilibrium on the amount of one of the ions is reached, a general system malfunction (denoted as $death = 1$) is established. As the other processes, the structure of this control process makes it possible the inclusion of additional features. Process $Start$, which receives a group of six parameters (denoted as $\sigma_{1...6}$), sets up the variables used in the model. Figure 3 shows a fragment of the complete model.

Remarkably, our models can be parametrized with actual quantitative values extracted from experimentation. Indeed, ion concentrations depend on *parameters* which make it more accurate; more detailed models involving other biological components (such as, e.g., the electrochemical gradients governing the dynamics of the passive transport) would then require the inclusion of more sophisticated numerical parameters. In this sense, considering a constraint system over real numbers would not only allow to include such parameters but also would allow to perform analyzes at different levels of detail.

Verifying the SP-pump

We now briefly describe how a non-trivial biological property can be verified over the sketched ntcc model of the SP-pump. Assume an *inhibition process* over the SP-pump that is enforced by a malicious drug that

$$
\begin{aligned}
NaKPump &\stackrel{\text{def}}{=} \mathbf{local}\ Na_I, Na_O, K_I, K_O, Alpha, ADP, Pump, OPump\ \mathbf{in} \\
&\qquad Start(\sigma_{1\ldots6}) \parallel ActiveTrans \parallel PassiveTrans \parallel Control \\
System &\stackrel{\text{def}}{=} NaKPump \parallel Environment
\end{aligned}
$$

Figure 3: Integrated ntcc model for the Sodium-Potassium pump (Fragment)

is present in the environment surrounding the pump. The goal of this drug is to take control of the alpha subunit, thus preventing the phosphate from inducing a conformational change in the pump. Such an this obstruction will lead to a complete inhibition of the active transport mechanism of the pump. We express this in our model by specifying the $Environment$ process as follows:

$$
Environment \stackrel{\text{def}}{=} Drug \star_{[m,n]}\ \mathbf{when}\ Alpha = \texttt{free}\ \mathbf{do}\ !Alpha := \texttt{null} \qquad (n > m) \qquad (1)
$$

It is easy to see that the actual time unit where $Drug$ will be active is undetermined, because of the uncertainty induced by the $\star$ operator. Notice that we are focusing on the drug-related part of $Environment$: other aspects of it could be easily specified.

Clearly, by inhibiting the active transport component of the pump, the cell will reach an equilibrium between the internal and external Sodium concentrations. Such an *irreversible* equilibrium causes the death of the cell and will occur in an undetermined future. These facts suggest us the following assertion to be verified:

$$
NaKPump \parallel Drug \vdash \Diamond\Box\, death = 1 \qquad (2)
$$

where $death = 1$ represents the death of the cell. Intuitively, we want to formally verify that in the presence of the drug described above the cell will die in an undetermined future, with no chance of returning to a previous state.

In [12] we use the inference system of ntcc to derive a proof for (2). Informally, the idea is to restrict the attention to the interaction among $Control$, $PassiveNa$ and $Drug$. Due to the absence of the active transport mechanism the passive transport will introduce sodium ions into the cell until reaching an equilibrium (i.e., $Na_I = Na_O$). Once that occurs, $Control$ (that has been awaiting the equilibrium) emits $equilNa = 1$ to the environment. Such a signal is enough to determine the death of the cell.

4.2 Genetic Regulatory Networks

Here we discuss how genetic regulatory networks (GRNs) can be modeled in ntcc. We propose a group of "building blocks" for modeling: each block represents a particular behavior that is frequent in GRNs. Some of these blocks are *generic processes* that can be parametrized according to the specific GRN, while others are *templates* that give guidelines on how to define actual ntcc processes. They have been used in [3, 2] to model and simulate regulation processes (repression and induction) of the *lac operon*, a genetic cluster that participates in the the transport and metabolism of lactose in bacteria such as the *E. Coli*.

Building Blocks for Modeling GRNs

GRNs are one of the most studied systems, mainly because of its importance at the cellular level. They control (or regulate) cellular processes according to the information provided by the ADN of each organism. At the molecular level, GRNs depend on many factors which make them particularly difficult to understand. Finding concise mathematical models describing behavior of GRNs is challenging as they are composed of elements that can be related to both discrete and continuous systems. In spite of this, it is possible to abstract some features that are common to GRNs at the molecular level. We now describe ntcc models for such features as *blocks* that might help to better formalize GRNs.

Continuity Regulation in GRNs is determined by the concentration levels of different biological entities along time. This motivates to consider two different kinds of continuity: persistence in the values of the variables and continuous time.

To model persistence of a single variable it is easy to think in a process $State_i$ that, for a variable m_i, explicitly transfers the current value of m_i to the next time unit. More precisely, the idea is, in the current time unit, to schedule a process $State_i(v_i)$ that will set the variable m_i' (which represents the value of m_i in the previous time unit) with v_i, the current value of m_i. This idea can be extended to a group of values in a straightforward manner:

$$State(\rho_1, \ldots, \rho_n) \stackrel{\text{def}}{=} \prod_{i \in I} (\mathbf{tell}(m_i' = \rho_i) \parallel \mathbf{next}\ (State_i(\rho_i)))$$

where I is the set of indexes of variables in the biological system and ρ_i is the current value of m_i. $State$ can be used to configure system simulations with parameters coming from actual biological measurements.

Temporal continuity is achieved by considering many `ntcc` time units as "samples" of one system unit:

$$Time_{Dt}(t) \stackrel{\text{def}}{=} \mathbf{tell}(Ts = t) \parallel \mathbf{next}\ (Time(t + Dt))$$

where Ts is the *continuous* time value of the system in the current time unit. Constant Dt represents the *resolution* of the system: it gives an idea of how fine the sampling is. As such, we can expect a trade-off between precision and efficiency: lower values of Dt give better approximations of real continuous systems but will demand more resources in system simulations. Process $Dynamic$ below can represent the continuous behavior of the whole system.

$$Dynamic \stackrel{\text{def}}{=} State(\rho_1, ..., \rho_n) \parallel Time_{Dt}(0.0)$$

Molecular Events Molecular systems involve several events that have to be considered, such as, e.g., the detection of when a group of molecules interacts with others or performs a specific task. We shall use discrete variables to indicate either presence or absence of molecular events in models. Such variables will be called *signaling variables*. The following is a generic `ntcc` process representing molecular behavior:

$$Signal \stackrel{\text{def}}{=} ! \prod_{e \in E,\ svar \in S} (\mathbf{when}\ e\ \mathbf{do}\ \mathbf{next}\ (\mathbf{tell}(svar = 1)) \parallel \mathbf{unless}\ e\ \mathbf{next}\ \mathbf{tell}(svar = 0))$$

where E is the set of constraints expressing molecular events and S the set of signaling variables in the system. Some readers might relate this process with an if-then-else construct. Nevertheless, $Signal$ provides a more sophisticated behavior as it can reason about *absence of information* on the conditions in E.

Regulation and status values Most of the processes used to represent dynamic behavior of biological entities share a similar structure. They can be modeled as processes controlled by signaling variables. The parametric process $Regulate_i$ models the behavior of an entity i which is under the control of a signaling variable $svar$. The value of $svar$ determines the execution of either P_i or N_i; this is represented as an exclusive choice.

$$Regulate_i(svar, P_i, N_i) \stackrel{\text{def}}{=} \mathbf{when}\ svar = 1\ \mathbf{do}\ P_i\ +\ \mathbf{when}\ svar = 0\ \mathbf{do}\ N_i$$

To model *status* (or level) of gene transcription, we use process $Status_i$ below as a *template* to define a wide variety of situations in which we want to determine particular conditions in/of a biological entity.

$$Status_i \stackrel{\text{def}}{=} ! \ (\ (\sum_{c \in C} \mathbf{when}\ cond_c\ \mathbf{do}\ \mathbf{next}\ (\mathbf{tell}(m_i = fc_i(m_i')))) \ \parallel \mathbf{unless} \bigvee_{c' \in C} cond_{c'}\ \mathbf{next}\ \mathbf{tell}(m_i = m_i'))$$

The above process assumes that conditions for changes in the status are indexed by the set C, so for two different i, j, $cond_i$ and $cond_j$ are two different conditions. The new value is defined by a control function fc_i. When no condition for change holds, the state of the system remains unchanged in the next time unit.

Genes Process Gen_x below is a parametric specification representing the structure and behavior of a single gene. It is defined using the generic process $Regulate_i$ and the template $Status_i$. The considered parameters represent the degradation and production rates of mRNAs as well as the proteins produced in the transcription and translation of a gene. We consider three entities: level of transcription and concentration of both mRNAs and proteins produced by the gene.

$$
\begin{array}{rcl}
GenStatus_i & \stackrel{\text{def}}{=} & !\ (\ (\ \mathbf{when}\ tbegin = 1 \wedge tend = 0\ \mathbf{do}\ \ \mathbf{next}\ (\mathbf{tell}(m_i = m_i' + 1))\ + \\
 & & \quad \mathbf{when}\ tbegin = 0 \wedge tend = 1\ \mathbf{do}\ \ \mathbf{next}\ (\mathbf{tell}(m_i = m_i' - 1))\ )\ \parallel \\
 & & \quad \mathbf{unless}\ \ tbegin \neq tend\ \ \mathbf{next}\ \ \ \mathbf{tell}(m_i = m_i')\ ) \\
MRNA_j(p_j, d_j) & \stackrel{\text{def}}{=} & Regulate_j(tbegin, \mathbf{next}\ (\mathbf{tell}(m_j = m_j' + p_j - Dt \times (d_j \times m_j'))), \\
 & & \quad \mathbf{next}\ (\mathbf{tell}(m_j = m_j' - Dt \times (d_j \times m_j')))) \\
PROTEIN_k(p_k, d_k) & \stackrel{\text{def}}{=} & Regulate_k(mrnah, \mathbf{next}\ (\mathbf{tell}(m_k = m_k' + Dt \times (p_k \times m_j' - d_k \times m_k'))), \\
 & & \quad \mathbf{next}\ (\mathbf{tell}(m_k = m_k' - Dt \times (d_k \times m_k')))) \\
Gen_x(p_j, d_j, p_k, d_k) & \stackrel{\text{def}}{=} & GenStatus_i \parallel !\ MRNA_j(p_j, d_j) \parallel !\ PROTEIN_k(p_k, d_k)
\end{array}
$$

In Gen_x, m_i, m_j and m_k are variables representing the status of gene expression, mRNA concentration and protein concentration, respectively. Moreover, d_j and d_k represent the rate of molecular degradation of mRNAs and proteins, respectively. The production rate of these entities is determined by the constants p_j and p_k and by two signaling variables $tbegin$ and $tend$. These denote the starting and ending time of RNA polymerase gene transcription. Signaling variable $mrnah$ is used to indicate when the mRNA concentration is "high enough" to start protein translation.

In order to model when RNA polymerase is placed between two genes an additional process is required. Such a process should control when each gene starts and finishes transcription. In [3] this process is modeled using the $Status_i$ template.

4.3 Modeling Biological Mutations

In this example we are interested in modeling the control system of a GRN. Below we define three `ntcc` processes: $StartControl$, $MutatedGene$ and $WildGene$. The first process indicates the number of molecules interacting with the control region at the start of the study of the system. The second one defines the system behavior under mutated conditions. The last one represents the system behavior in wild or normal conditions. Variable x represents the cellular concentration of molecules interacting with the control region of the set of genes.

$$
\begin{array}{rcl}
StartControl & \stackrel{\text{def}}{=} & \mathbf{tell}(x = n) \\
MutatedGene & \stackrel{\text{def}}{=} & \star\,!\,(\mathbf{tell}(mut = 1) \parallel \mathbf{next}\,(\mathbf{tell}(x = f_m))) \\
WildGene & \stackrel{\text{def}}{=} & !\ \mathbf{unless}\ \ mut = 1\ \ \mathbf{next}\ \ \ \mathbf{tell}(x = f_w) \\
ControlRegion & \stackrel{\text{def}}{=} & Start \parallel MutatedGene \parallel WildGene
\end{array}
$$

In the above definitions, process $MutatedGene$ establishes that a mutation will eventually occur in the gene in an undetermined future time unit and, as a consequence, the behavior of the system will be defined by the constraint $x = f_m$, where f_m is a function determining an incorrect behavior in the gene control region. In addition, process $WildGene$ states that the behavior of the control region is represented by the constraint $x = f_w$ unless the mutation occur (which is represented by constraint $mut = 1$). Function f_w represents the behavior of the system in wild conditions. Figure 4, obtained using `ntccSim`, illustrates the behavior of the system parametrized with value $n = 0$.

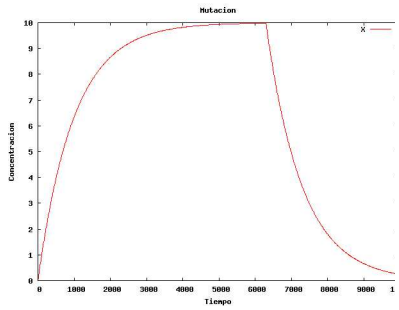


Figure 4: Molecular concentration in a DNA region of a mutated gene

A Complementary Proof In this section we will verify a system property using the inference system associated with `ntcc`. As a case of study, we will verify that when the mutation occur, variable x will be determined only by function f_m. Formally, we wish to verify the following formula:

$$ControlRegion \vdash \Diamond\Box x = f_m$$

The formulas for processes $StartControl$, $MutatedGene$ and $WildGene$ are:

$$\begin{array}{lll} StartControl & \vdash & x = n \\ MutatedGene & \vdash & \Diamond\Box(mut = 1 \dot{\wedge} \circ x = f_m) \\ WildGene & \vdash & \Box(mut = 1 \dot{\vee} \circ x = f_w) \end{array}$$

$$\dfrac{\dfrac{}{StartControl \vdash x = n}\ \mathsf{LTELL} \quad \dfrac{}{MutatedGene \vdash \Diamond\Box(mut = 1 \dot{\wedge} \circ x = f_m)}\ \mathsf{LRULES1}}{StartControl \parallel MutatedGene \vdash\ (\ x = n\ )\ \dot{\wedge}\ (\ \Diamond\Box(mut = 1 \dot{\wedge} \circ x = f_m)\ )}\ \mathsf{LPAR}$$

where LRULES1 denotes the systematic application of rules LSTAR, LREP, LPAR, LNEXT and LTELL of the proof system over process $MutatedGene$. For the sake of space, we assume the following abbreviations: $SC = StartControl$ and $MG = MutatedGene$.

$$\dfrac{\dfrac{}{WildGene \vdash\ (\ \Box(mut = 1 \dot{\vee} \circ x = f_w)\ )}\ \mathsf{LRULES2} \quad \dfrac{}{SC \parallel MG \vdash\ (\ x = n\ )\ \dot{\wedge}\ (\ \Diamond\Box(mut = 1 \dot{\wedge} \circ x = f_m)\ ))}}{WildGene \parallel SC \parallel MG \vdash\ (\ \Box(mut = 1 \dot{\vee} \circ x = f_w)\ )\ \dot{\wedge}\ (\ x = n\ )\ \dot{\wedge}\ (\ \Diamond\Box(mut = 1 \dot{\wedge} \circ x = f_m)\ )}\ \mathsf{LPAR}$$

where LRULES2 represents the application of rules LREP, LUNL and LTELL over process $WildGene$. Finally, we can perform the following deduction:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{ControlRegion \vdash\ (\ \Box\ (mut = 1 \dot{\vee} \circ x = f_w)\ )\ \dot{\wedge}\ (\ x = n\ )\ \dot{\wedge}\ (\ \Diamond\Box\ (mut = 1 \dot{\wedge} \circ x = f_m)\ )}\ \mathsf{LCONS}}{ControlRegion \vdash \Box\ (mut = 1 \dot{\vee} \circ x = f_w) \dot{\wedge} \Diamond\Box\ (mut = 1 \dot{\wedge} \circ x = f_m)}\ \mathsf{LCONS}}{ControlRegion \vdash \Diamond\Box\ (\ \ (mut = 1 \dot{\vee} \circ x = f_w) \dot{\wedge} (mut = 1 \dot{\wedge} \circ x = f_m)\ )}\ \mathsf{LCONS}}{ControlRegion \vdash \Diamond\Box\ (\ mut = 1 \dot{\wedge} (mut = 1 \dot{\wedge} \circ x = f_m)\ )}\ \mathsf{LCONS}}{ControlRegion \vdash \Diamond\Box \circ x = f_m}\ \mathsf{LCONS}}{ControlRegion \vdash \Diamond\Box x = f_m}\ \mathsf{LCONS}$$

The above logical expression verify that the constraint $x = f_m$ will define the behavior of the system in an undetermined future time, and that this behavior will continue forever.

We have shown how the behavior of a system can be analyzed by two formal ways: (i) by following the steps of the operational semantics in a mechanical way, using ntccSim (Figure 4) and (ii) by verifying temporal properties using the ntcc inference system. A remarkable aspect to consider here is that it is possible that we may not see the mutation by simulations, since this could occur in a very long time. As a consequence, in this case the logical proof can be regarded as being more effective, as it can reveal the actual behavior of the system.

5 Related Work

Some of the main representative calculi within the so-called language approach for systems biology are the π-calculus [22, 23], BioAmbients [21], the Brane calculus [8], Beta binders [20] and the κ-calculus [9]. The use of these calculi as as description languages for Biology has been studied in recent years and, as mentioned in the introduction, little work has been done on relating them with logic-based reasoning techniques. Some of such works have explored the use of constraints and/or logic in the biological context, see, e.g., [11, 5, 6, 1]. Two of them ([5, 6]) are most related to our approach and deserve special mention. We review them separately.

Stochastic CCP Stochastic CCP (sCCP) [5] is an untimed, stochastic extension of the ccp model. The main difference wrt the original model proposed in [27] is the addition of a λ function to ask and tell operations as well as to procedure calls. The intuitive meaning of this function is twofold. In fact, it can be understood either as a priority within a probability distribution or as the speed associated with performing each operation. From a practical perspective, there is an interpreter of sCCP processes, built in SICStus Prolog, that allows for simulation of biological systems.

In order to model biochemical networks, the work in [5] offers parameterizable processes to describe reversible and irreversible reactions as well as reactions described by Michaelis-Mentel and Hill equations. The definition of similar processes in ntcc this is also possible. Moreover, to model genetic regulatory networks, three basic processes (or *logical gates*) are proposed to model regulation. More precisely, processes *pos_gate*, *neg_gate* and *null_gate*, intended to model positive, negative and absence of regulation, respectively, are proposed. This kind of sCCP processes can be easily modeled in ntcc.

Clearly, the use of stochastic parameters is the main difference between sCCP and ntcc. We have already started to work on equipping ntcc with probabilistic/stochastic constructs (see Section 6). We feel that the combination of probabilistic behavior with the discussed advantages of ntcc in the biological context (time, partial information, logic reasoning techniques) will constitute a strong framework where biological systems can be better studied.

Temporal Logic with Constraints The works [6, 7] propose BIOCHAM, a biochemical abstract machine. In BIOCHAM biological systems are modeled using a rule-based language. This approach is, according to the authors, more natural to the biologists and well-suited for applying model checking techniques. This is perhaps the main difference wrt our approach, as processes in ntcc have a natural relationship with the temporal logic associated to the language. Furthermore, we think that the explicit time representation inherent to ntcc can, in combination with the non-deterministic and asynchronous constructs, be intuitive enough for experts when describing the behavior of (possibly partially known) biological systems.

Reasoning techniques include three independent semantic structures (each one with an associated logic), which are used depending on the desired level of detail. The simplest semantics is a *boolean* one that associates a boolean variable to each biological entity, with the possibility of checking *qualitative* properties using Computational Tree Logic (CTL). In the *concentration* semantics each entity is associated to a real number representing its concentration. Reaction rules are interpreted as kinetic rules and a fragment of LTL is used for verification. Finally, in the *stochastic* semantics an integer is used to model the number of molecules of each entity in the system. Notice how for each level of abstraction there is a different meaning for the modeling language and different verification approaches. We believe that by the appropriate use of

constraint systems in the description of systems, analysis at several levels of detail are possible, preserving the *same unified framework*.

6 Concluding Remarks and Future Work

In this paper we have shown how ntcc, a timed, non-deterministic process calculus based on constraints, can be convenient for the analysis of different kinds of biological systems. We have seen how the interplay of the operational and logic perspectives of processes —a distinctive feature of ccp languages— serves as a unified framework upon which expressive models of biological systems can be described, observed and, unlike most similar works, verified using a temporal logic.

The discussed biological systems serve to illustrate the advantages of using ntcc in the biological context. In fact, ntcc allows to take advantage of the natural use of *processes* as independent agents to model biological entities, *discrete time* constructs as flexible mechanisms to describe dynamic properties of systems, *constraints* as a way of representing incomplete information about the state of a system (i.e., partial *quantitative* information), and *asynchronous and non-deterministic* constructs to formally model unpredictable behavior in the evolution of a system (i.e., partial *behavioral* information). Moreover, these advantages in modeling are complemented by both practical and theoretical opportunities for simulating and verifying biological models. On the one hand, it is possible to run ntcc specifications in ntccSim in order to know a possible execution path showing the behavior of a system, given a set of particular conditions (e.g., initial number of molecules in a system). On the other hand, the use of an LTL inference system to prove temporal properties about ntcc models allows to discover non-trivial behavior patterns, including those encompassing asynchronous and non-deterministic nature. This tight relationship between operational and logic reasoning tools is rarely seen in other formalisms, even in those also based on the ccp model.

All these appealing features certainly motivate us to further work on the applications of ntcc to the biological context. A current work direction pertains to the use of quantitative information in models of biological systems. Particularly important is the inclusion of probabilistic/stochastic information both in ntcc models and ntccSim. We have already obtained some preliminary results. In fact, in [17] a version of ntcc in which the signature of the constraint system is extended with a probability function is proposed. Intuitively, the role of such a function is to return "true" or "false", taking a real number as a parameter. This adds a significant flexibility to process definitions, as one could devise processes that are executed depending on the outcome of such a function. The advantages of using this extended language in the biological context were described in [18], where cooperativity in a genetic regulation network is formally studied. Moreover, we count with preliminary results on the design of a *probabilistic* extension of ntcc with probabilistic choice. We expect to refine these theoretical extensions by modeling, simulating and verifying more complex biological systems than the ones analyzed so far.

From a more practical point of view, the development of efficient mechanisms for including ordinary differential equations (ODEs) in models and simulations is also compulsory. Although we have defined some encodings of ODEs using ntcc, we plan to implement a constraint system over ODEs in Mozart. Such a system, in combination with the existing constraint systems over real intervals and finite domains, will allow to take advantage of the knowledge currently held by biologists about the structure and behavior of molecular systems, and consequently, to contribute to fill the gap that prevents computer scientists from straightforwardly using some well studied models of biological networks.

References

[1] M. Antoniotti, C. Piazza, A. Policriti, M. Simeoni, and B. Mishra. Taming the complexity of biochemical models through bisimulation and collapsing: theory and practice. *Theor. Comput. Sci.*, 325(1):45–67, 2004.

[2] A. Arbeláez and J. Gutiérrez. Estudio Exploratorio de la Aplicación de la Programación Concurrente por Restricciones en Bioinformática (Applying Concurrent Constraint Programming in Bioinformatics: An Exploratory Study). BSc Thesis – Engineering Degree in Computer Science, Pontificia Universidad Javeriana - Cali (Colombia)., 2006.

[3] A. Arbeláez, J. Gutiérrez, C. Olarte, and C. Rueda. A generic framework to model, simulate and verify genetic regulatory networks. In *Proc. of 32nd Latin-American Conference on Informatics (CLEI 2006)*, 2006.

[4] AVISPA Research Group. ntccSim: A simulation tool for timed concurrent processes, 2006. More information available at `http://avispa.puj.edu.co`.

[5] L. Bortolussi and A. Policriti. Modelling Biological Systems in Stochastic Concurrent Constraint Programming. In *Proc. of WCB06 - Workshop on Constraint-Based Methods in Bioinformatics*, 2006.

[6] L. Calzone, N. Chabrier-Rivier, F. Fages, and S. Soliman. Machine learning biochemical networks from temporal logic properties. *Transactions on Computational Systems Biology*, 2006. CMSB'05 Special Issue (to appear).

[7] L. Calzone, F. Fages, and S. Soliman. BIOCHAM: An Environment for Modeling Biological Systems and Formalizing Experimental Knowledge. *Bioinformatics*, 22:1805–1807, 2006.

[8] L. Cardelli. Brane Calculi. In Danos and Schächter [10], pages 257–278.

[9] V. Danos and C. Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.

[10] V. Danos and V. Schächter, editors. *Computational Methods in Systems Biology, International Conference CMSB 2004, Paris, France, May 26-28, 2004, Revised Selected Papers*, volume 3082 of *LNCS*. Springer, 2005.

[11] D. Eveillard, D. Ropers, H. de Jong, C. Branlant, and A. Bockmayr. A multi-scale constraint programming model of alternative splicing regulation. *Theor. Comput. Sci.*, 325(1):3–24, 2004.

[12] J. Gutiérrez, J. A. Pérez, C. Rueda, and F. D. Valencia. Timed Concurrent Constraint Programming for Analysing Biological Systems. In *Proc. of the Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'06)*, 2006. To appear in the ENTCS (Electronic Notes in Theoretical Computer Science) series.

[13] P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, Implementation, and Evaluation of the Constraint Language cc(FD). In *Constraint Programming*, volume 910 of *LNCS*, pages 293–316. Springer, 1994.

[14] H. Kitano, editor. *Foundations of Systems Biology*. MIT Press, 2001.

[15] H. Kitano. Systems Biology: A Brief Overview. *Science*, 295:1662–1664, 2002.

[16] M. Nielsen, C. Palamidessi, and F. Valencia. Temporal concurrent constraint programming: Denotation, logic and applications. *Nordic Journal of Computing*, 9:145–188, 2002.

[17] C. Olarte and C. Rueda. A Stochastic Non-deterministic Temporal Concurrent Constraint Calculus. In *Proc. of International Conference of the Chilean Computer Science Society (SCCC 2005)*. IEEE-CS, 2005.

[18] C. Olarte and C. Rueda. Using stochastic ntcc to model biological systems. In *Proc. of the 31st Latin American Conference on Informatics (CLEI'05)*, 2005.

[19] D. Prandi, C. Priami, and P. Quaglia. Process calculi in a biological context. *Bulletin of the EATCS*, February 2005.

[20] C. Priami and P. Quaglia. Beta Binders for Biological Interactions. In Danos and Schächter [10], pages 20–33.

[21] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro. Bioambients: an abstraction for biological compartments. *Theor. Comput. Sci.*, 325(1):141–167, 2004.

[22] A. Regev and E. Shapiro. Cells as Computation. *Nature*, 419:343, September 2002.

[23] A. Regev and E. Shapiro. *Modelling in Molecular Biology*, chapter The π-calculus as an abstraction for biomolecular systems, pages 219–266. Natural Computing Series. Springer, 2004.

[24] Aviv Regev, William Silverman, and Ehud Y. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Pacific Symposium on Biocomputing*, pages 459–470, 2001.

[25] V. Saraswat. *Concurrent Constraint Programming*. The MIT Press, 1993.

[26] V. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 71–80. IEEE, 1994.

[27] V. Saraswat, M. Rinard, and P. Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91*, pages 333–352, Jan 1991.

[28] G. Scheiner-Bobis. The sodium pump: Its molecular properties and mechanics of ion transport. *Euro. J. Biochem.*, 269:2424–2433, 2002.

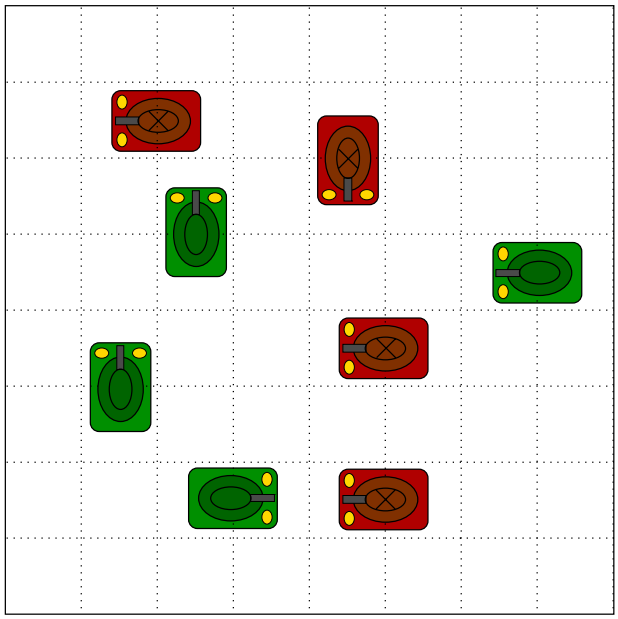
Cool War

Paolo Baldan

November 6, 2006

In year 6002 planet Earth has reached a high degree of civilization and war has essentially disappeared. Only a vestige of war survives in the way international controversies between countries are resolved: The two opponent countries, say C1 and C2, set up a square battlefield and they turn in placing their tanks in the field. Tanks cannot overlap. The winner is the last country which is able to place a tank.

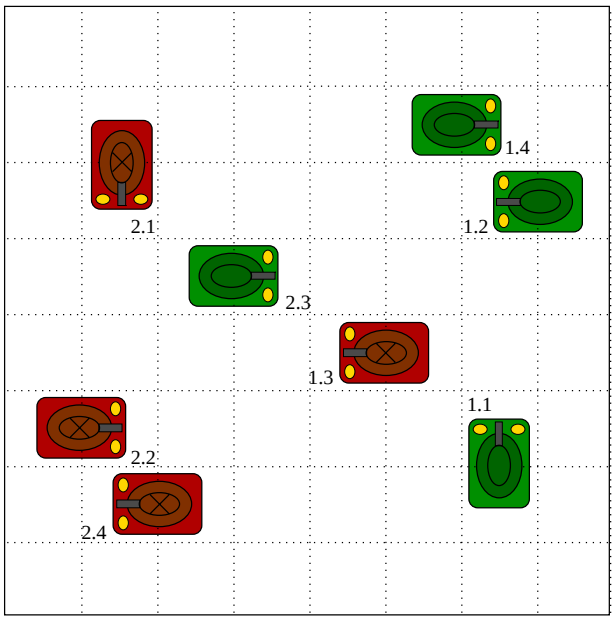
To make the process more disciplined, the battlefield is divided into $n \times n$ squares (as a kind of chessboard), where n is even, and each tank occupies two adjacent squares, horizontally in a row or vertically in a column. As an example, you can see below a possible configuration of the field after four turns. The tanks placed by C1 are green, while those of C2 are red (and they bring a cross on the top, in case you have a black and white printout).



The battle is normally very quiet ... the two opponent countries are quite nervous only when they have to decide who should start placing the first tank.

Q1. *Can you explain why? If C1 is the country that starts placing the tanks, does a winning strategy for C1 or for C2 exist?*

Answer: Yes, there is a winninig strategy for C2. If C1 places a tank at a certain position, say p, C2 always answer by placing a tank at a position p' which is symmetric to p with respect to the center of the field. Since this maintains the symmetry of the field, whenever C1 is able to place a tank, also C2 will be able to do so, and thus C2 will surely be the winner. A possible configuration of the field after four steps is exemplified below. Tanks are numbered as $x.y$ where x stands for the owner and y is a progressive number denoting at which turn the tank has been placed.



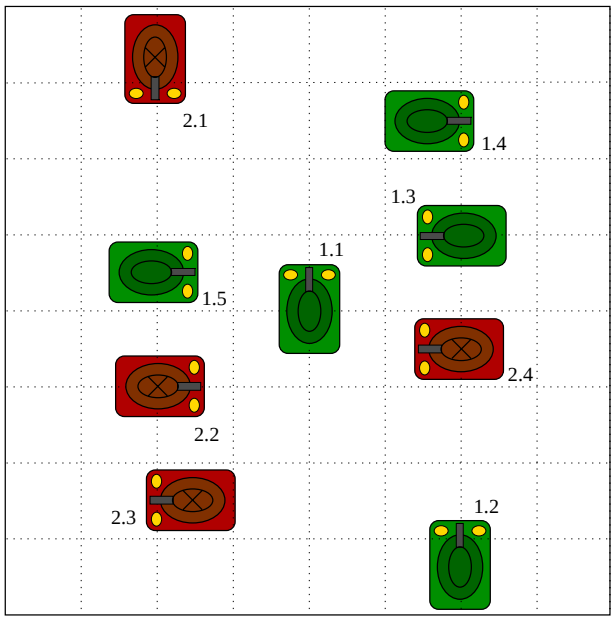
Note that the above argument applies to the case of "ordinary" tanks, which need a free path to access their final position, and also to flying tanks, which can be placed in any free position in the field, even if it not accessible.

Once the existence of a winning strategy is discovered, the regulation is drastically changed: the battlefield is no more organised in a grid, and now the tanks can be placed in any position, with any orientation, but still avoiding any overlapping.

Q2. *Do you think that this improves the situation?*

Answer: Not really. Now there is a winning strategy for C1. In fact, C1 can start placing the tank at the center of the board. Then, the previously illustrated strategy can be reused: at any turn C1 will play symmetrically with respect to

the move of C2 and thus C1 will be the winner. A possible configuration of the field, after five moves of $C1$ and four moves of $C2$ can be found below.



Conference Report
Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS'06)

Axel Polleres
Universidad Rey Juan Carlos
Spain

Editor: Enrico Pontelli

This report gives a brief overview of the topics and results of the workshop on ``Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS 2006)", which took place for the first time on August 16th 2006, in Seattle, WA, USA, co-located with ICLP. Besides the usual presentation of scientific papers, we adopted a methodology called Open-Space in order to facilitate a more interactive workshop atmosphere, which, although not yet very common for scientific workshops, proved to be very stimulating and even led to the concrete planning of collaborations among the workshop participants.

The advent of the Semantic Web promises machine-readable semantics and a machine-processable next generation of the Web. The first step in this direction is the annotation of static data on the Web by machine processable information about knowledge and its structure by means of Ontologies. The next step in this direction is the annotation of dynamic applications and services invocable over the Web, in order to facilitate automation of discovery, selection, and composition of semantically described services and data sources on the Web by intelligent methods; this is called Semantic Web Services. Many workshops and conferences have been and are being dedicated to these promising areas. They mostly deal with generic topics and bringing together people from a broad variety of research fields with different understandings of the topic. The plethora of these workshops and conferences makes it hard to keep track of the various approaches of a particular technology, such as declarative logic programming in our case. So, the goal of ALPSWS was somewhat different: We aimed at

advancement of specific applications of Logic Programming as a paradigm for declarative knowledge representation and reasoning for the Web. The idea was to bring together the impressive body of work related to applications of Logic Programming to Semantic Web and Semantic Web Services in order to bring together people from different sub-disciplines of LP and focus on technological solutions and applications from LP to the problems of the Web.

Co-locating the event with one of the major Semantic Web conferences (ISWC, ESWC, ASWC) seemed like ``carrying OWLs to Athens'' (BTW, this year's ISWC indeed takes place in Athens... Georgia, USA, however ;-)), so we decided to co-locate the workshop with ICLP in Seattle with the idea to further promote research in this interesting application field in the LP community.

The workshop took place on Wednesday, August 16th; it lasted a full day and was divided into three sessions. While the first block was dedicated to papers on Semantic Web reasoning, the second block hosted contributions for applying LP to reasoning about Web services and Policies. A final block was dedicated to discussions around the topic of the workshop, adopting the so-called *Open-Space* methodology, which was started with a half an hour poster session in order to stimulate discussion.

From the seven accepted full papers (see **http://events.deri.at/alpsws2006/#program**), four treated core LP and Semantic Web reasoning issues, whereas three were mainly concerned with Web services. Among the five posters, which were
additionally presented, two were concerned with Web services, one with syntax, and we had two interesting contributions on OWL Reasoning using Prolog.

Concerning adopted LP paradigms, a tendency, which could also be observed at ICLP, was recognizable: Whereas most of the papers used traditional Prolog, a considerable number of contributions was dedicated to the recently emerging Answer Set Programming paradigm, where both paradigms proved valuable in their particular applications. Roughly speaking, Prolog-based approaches have proven valuable for implementations of Semantic Web and Semantic Web Services solutions for a variety of applications, as well as for building lightweight reasoners, whereas the ASP contributions focused more on the aspects of using LP itself as a KR formalism for the Web. It will be interesting to follow the ongoing development, not only in the particular field of Semantic Web, but in the LP community in general.

One of the most interesting parts of the workshop for us as the organizers was the

open-space discussion in the afternoon session. Open-Space is a methodology for joint agenda finding (see e.g. **http://en.wikipedia.org/wiki/Open_Space_Technology**) which facilitates dynamic discussions. It turned out that the methodology is well-suited for such a workshop. The interesting thing about this form of discussion is that you never know what happens, since the participants themselves decide about the items to be discussed, and several discussions are happening in parallel in the same room; people join and leave discussions absolutely voluntarily, but it (surprisingly?) still almost always works!

Six concrete topics were discussed throughout the Open-Space sessions which we describe (in a deliberately sketchy way) in the following. Not all discussions had a ``tangible'' outcome, but many interesting issues were raised which increased the common understanding of the problems in the area.

Open world reasoning vs. closed world reasoning. This topic was concerned with the different implications and needs for Semantic Web Reasoning. Much time was spent on reaching a common understanding of the distinction between "open" and "closed" in the context of the Web. E.g., negation as failure may be viewed as closed world reasoning on the one hand, but also be interpreted as simply reasoning by default, which is very natural for humans. OWL, as opposed to LP dialects advocates a strict open world reasoning. However, if the Web is to be the ``world's largest large database'', what are the implications, as database query languages (including SPARQL!) do have negation as failure? Closed world reasoning is very common in databases. Can scoped negation over closed Web documents or local closed world reasoning do the trick?

Ontologies and Web Services. This topic was concerned with the meaning of Ontologies as such, as well as the meaning of Ontologies for (Web) services. The focus, though, was on the former, including the discussion of different Representation Formalisms and different Ontologies. Other topics: What does it mean to ``harmonize'' different ontologies? How to do ``Web services search''? In how far do/can search engines make commitment to ``meaning''? Reuse as a crucial issue of ontologies; discussion of a concrete application area - eHealth - where standards with completely different backgrounds and communities exist.

Smart Search Engines. Challenges and open issues regarding smart search engines were discussed: How to reduce annotation costs which are a high entry barrier to the Semantic Web? Which technologies can be used? How to increase the benefit compared to current search engines? What \emph{output} do you expect from "smart" search engines? What is the meaning of correctness/ reliability/locality in search results (correctness vs. "google")? How can you

specify constraints?

Semantic Modeling of Web Services Inputs/Outputs. This slot was dedicated to the often overlooked problems of semantically modeling inputs and outputs of Web services. It is often not enough just to say that the input shall be an instance of an (OWL) ontology, because this raises the following question: What to send over the wire? CORBA and XML Schema both provide formalisms to check valid inputs, whereas this problem is to a large extent ignored/neglected in Semantic Web Services frameworks. It remains unclear how to set up a contract, when e.g. an instance of the class "Person" is requested as service input. What shall be sent by the requester? A person id? A name and social insurance number? It was agreed among the workshop participants to jointly work on a solution of this issue, defining a formal specification of semantically described input and output.

Joint project opportunities. Which possible funding lines for joint projects exist on EU level? How can US (or other non-EU) partners join? What are the most promising/challenging application areas (also from the perspective of funding possibilities)? Several participants showed interest in joint projects and first contacts have been made (and, in the meantime, have even led to the first joint proposals).

Next ALPSWS Fortunately, everyone agreed that we had a stimulating and useful workshop, which made positions, approaches, but also open issues and gaps clear. One of the main issues turned out to be the question how to bridge the gap between Semantic Web and Web services. LP methods are successful for both, but so far hardly in a unified framework: In this slot, we discussed where and when to have a next ALPSWS in order to further encourage cross-fertilization of these fields.

We further discussed how to keep the spirit to make the event a *work*shop instead of a mini-conference. The Open-Space discussions were agreed to be a perfect means to this end. Additionally/alternatively, possibilities such as a panel or different forms of presentations (e.g. 10min flash presentations plus a longer poster session for all papers) were discussed.

Summarizing, the first three of the Open-Space sessions rather served to achieve a higher level of mutual understanding, than leading to specific joint work, whereas the latter three already seem to have sparked concrete activities.

We plan to collect the outcome of these discussions on a separate Wiki page (which will be made accessible via the workshop Webpage (**http://events.deri.at/**

alpsws2006/) soon and hope that the participants, but also other interested people joining, will take up the defined action items and be able to present more concrete answers to the open questions mentioned above.

Proceedings including the full papers and abstracts of presented posters are available online in the CEUR Workshop Proceedings series, under Vol-196 (see **http://ceur-ws.org/Vol-196/**). Additionally, the two best papers will be published in a dedicated section of the upcoming special issue on "Logic Programming and the Web" in TPLP, see **http://www.dsi.unive.it/~tplp/**.

Hopefully, see you -- and awaiting your interesting contributions - at a possible continuation of this workshop!

Axel Polleres, Stefan Decker, Gopal Gupta, and Jos de Bruijn (the organizers)

Conference Report
2006 Colloquium on Implementation of Constraint Logic Programming Systems (CICLOPS'06)

Haifeng Guo
University of Nebraska at Omaha
USA

Editor: Enrico Pontelli

The Colloquium on Implementation of Constraint LOgic Programming Systems (CICLOPS 2006) took place on August 21, 2006 as a satellite workshop of ICLP'06 in Seattle, Washington. This workshop continued a tradition of a successful workshops on implementations of (Constraint) Logic Programming Systems since 1993.

The workshop's aim was to promote discussion and exchange of experience on the design, implementation, and optimization of logic and constraint (logic) programming systems, or systems intimately related to logic as a means to express computations.

This year we received eight papers which address various implementation topics including tabling, preferences, region-based memory management, action rules, deductive databases, inductive logic programming and linear logic. Interestingly, half of the papers are related to tabling systems. The CICLOPS'06 program commitee accepted all the papers for presentations. The following submissions were presented:

- *H. Guo, M. Liu*. Embedding Solution Preferences via Transformation
- *T. Soares, M. Ferreira, R. Rocha, N. Fonseca*. On Applying Deductive Databases to Inductive Logic Programming: a Performance Study
- *R. Rocha*. Efficient Support for Incomplete and Complete Tables in the YapTab Tabling System
- *B. Demoen, P-L. Nguyen*. Delay and events in the TOAM and the WAM

- *Q. Phan, G. Janssens*. Towards Region-based Memory Management for Mercury Programs
- *R. Troncon, B. Demoen, G. Janssens*. When tabling does not work
- *P. Costa, R. Rocha, M. Ferreira*. DBTAB: a Relational Storage Model for the YapTab Tabling System
- *L. Chrpa*. Linear Logic: Foundations, Applications and Implementations

Additionally, an invited talk, titled "AR (Actiona Rules): the language, implementation, and applications", was given by Dr. Neng-Fa Zhou from the City Univerity of New York at Brooklyn College.

On behalf of the organizers we would like to thank all the PC members for their timely reviews, all the authors for their paper contributions, and all the participants for their questions and discussions. Also, we wish to thank Dr. Neng-Fa Zhou for accepting our invitation and giving an interesting talk on action rules.

Conference Report
Software Verification and Validation (SVV'06)

Abhik Roychoudhury
National University of Singapore
Singapore

Zijiang Yang
Western Michigan University
USA

Editor: Enrico Pontelli

The fourth International Workshop on Software verification and Validation was held as an ICLP'06 workshop as part of the Federated Logic Conferences (FLoC) 2006. Since its inception in 2003 as an ICLP workshop, SVV has grown to be a forum for discussing combinations of various software validation methods (automated or manual, formal or informal). This year, we received 12 papers out of which 7 were selected for presentation by the Program Committee.

The workshop was held on August 21. It started with an invited presentation, *Automatic Termination Proofs for Systems-level Code*, by **Byron Cook** from Microsoft Research. In the talk Dr. Cook described recent advances in the area of automatic program termination analysis. In particular, he presented the development of several new automatic tools, called TERMINATOR and MUTANT, which implement new termination analysis algorithms.

The first presentation of accepted papers was given by **Ganna Zaks** from New York University. She presented the paper *"Translation Validation of Interprocedural Optimizations"*, co-authored with her advisor **Amir Pnueli**. Translation Validation is an approach of ensuring compilation correctness in which each compiler run is followed by a validation pass that proves that the target code of the program produced by the compiler is a correct translation of the source code. In their work, they extended the existing framework for translation validation of optimizing compilers to deal with procedural programs and define the notion of correct translation for reactive programs with intermediate inputs and outputs.

The second presentation was given by Dr. **Luke Simon** from the University of Texas at Dallas. Two of his co-authors, **Ajay Mallya** and **Ajay Bansal,** also

attended the workshop. In the talk paper Dr. Simon presented the theory and practice of co-logic programming, a paradigm that combines both inductive and coinductive logic programming. An outline of a prototype implementation realized by modifying YAP Prolog's engine at the WAM level was described.

After the lunch break, **Aleks Zaks** from New York University presented his paper on Using Range Analysis for Software Verification, co-authored with seven other researchers from NEC, Western Michigan University, and Real Intent. Two of his co-authors, Drs. **Ilya Shlyakhter** and **Zijiang Yang**, attended the workshop**.** The main contributions of their paper are two light-weight range analysis techniques for determining lower and upper bounds for program variables, and their application in improving various software model checking techniques.

Dr. **Elvira Albert** from Complutense University of Madrid presented his paper next on Java Bytecode Verification using Analysis and Transformation of Logic Programs. The aim of their work is to automatically transfer the power of analysis tools for LP to the analysis and verification of Java bytecode. In order to achieve our goal, they rely on well-known techniques for meta- programming and program specialization.

The fifth presentation was given by **Stefano Tonetta** from University of Lugano. His co-author, Dr. **Natasha Sharygina** from UNISI, was present during the presentation. Mr. Tonetta presented a uniform framework for predicate abstraction approximation. The mapping among various abstraction techniques provides a conceptual basis for the development of new algorithms.

After a short coffee break, the last session started at 4pm. Dr. **Burkhart Wolff** from ETH Zurich presented a package for extensible object-oriented data models with an application to imp++. They implemented a datatype package that enabled the shallow embedding technique to object-oriented specification and programming languages.

The last presentation was given by Mr. **Ossami** from LORIA. Mr. Ossami and his two coauthors proposed a process model for the development of formal and semi-formal specifications based on the notions of multi-view state and of development operators.

Besides the invited speaker and authors of accepted papers mentioned above, the attendees of SVV include Dr. Hill from University of Leeds, Dr. Koprowski from Eindhoven University of Technology, Dr. Lukácsy from Budapest University

of Technology and Economics, and Dr. Veith from TU Munich. The workshop was adjourned at 5:30pm.

Conference Report
2006 Workshop on Constraint-based Methods for Bioinformatics
(WCB'06)

Agostino Dovier
University of Udine
Italy

Alessandro Dal Palù
University of Parma
Italy

Sebastian Will
University of Freiburg
Germany

Editor: Enrico Pontelli

Bioinformatics is a challenging area of research where every serious contribution can have thousands of positive effects in medicine, agriculture, or industry.

Most of the typical problems can be effectively formulated by using declarative languages and constraints. Constraint on finite domains (and on reals) are applied for predicting spatial conformation of polymers, concurrent constraint programming can be used for simulations of biological systems, and finally constraints on strings are employed for the analysis of DNA sequences.

The WCB06 workshop was organized with the aim of sharing new theoretical and practical results in the area and to discuss whether there are new challenging problems for the declarative programming and constraint community. The worskhop is the successor of the workshops *Constraints and Bioinformatics/ Biocomputing*, co-located with CP'97 and CP'98, and of the workshop WCB'05, co-located with ICLP 2005.

The workshop benefited by the excellent invited talk of Francois Fages about *"Using temporal logics with constraints to express biological properties of cell processes"* and by the presentation of 7 contributed papers ranging from protein folding to system biology, from suffix arrays to supertree construction. The interest of the constraint community on this theme is witnessed by the number of participants (35) although the workshop ran in parallel with other extremely interesting workshops.

An extended summary of the workshop will appear as a chapter of the book

"Trends in Constraint Programming" edited by Frèdèric Benhamou, Narendra Jussien, and Barry O'Sullivan, to be published by Hermes Science. Moreover, a special issue of *Constraints* Journal on the topics of the workshop will be announced in the following weeks.

During the final discussion, we decided to try to colocate the next edition with ICLP06 in Porto, where we hope to meet the Logic Programming community that can give a strong contribute to this research area.

Other information, pictures from the workshop, and the proceedings can be found in the workshop web-site **http://www.dimi.uniud.it/dovier/WCB06**.

We conclude by acknowledging the PC members, the external referees, all the participants, and the CP workshop chair Barry O'Sullivan.

Conference Report
Logic-based Methods in Programming Environments
(WLPE'06)

Wim Vanhoof
University of Namur
Belgium

Editor: Enrico Pontelli

This year's edition of *WLPE*, the *Workshop on Logic-based Methods in Programming Environments*, took place high above downtown Seattle, on the 30th floor of the welcoming Sheraton and Towers hotel. The aim of the workshop was to provide an informal meeting for researchers working on (logic-based) methods and tools that support the development and analysis of programs.

The workshop attracted 12 submissions of which the program committee decided to accept 9 for presentation, resulting in quite a busy program for a single afternoon. The presentations were grouped around 3 major themes.

First, we had a session on *debugging and tracing*, with presentations by Remko Tron•on (*a delta debugger for ILP query execution*), Pierre Deransart ({*on using tracer driver for external dynamic process observation*), and Bharat Jayaraman (*JavaTA: a logic-based debugger for Java*).

Then followed a session on program analysis, with talks by Manuel Hermenegildo (*towards execution time estimation for logic programs via static analysis and profiling*), Michael Hanus (*CurryBrowser: a generic analysis environment for Curry programs*), German Puebla (*some issues on incremental abstraction-carrying code*), and Wim Vanhoof (*fingerprinting logic programs*).

Finally, we had a session on tools, with presentations by Siddharth Chitnis (*ExSched: solving constraint satisfaction problems with the spreadsheet paradigm*) and John Gallagher (*a web-based tool combining different type analyses*).

Among the participants to the workshop, there were 16 who registered oficially for WLPE, coming from 9 different countries. The workshop itself was an informal and lively event, with high-quality presentations being interleaved with an appropriate amount of discussion.

Although it was a long and intense afternoon, it was first of all a very pleasant and instructive one.

Conference Report
Italian Conference on Computational Logic (CILC'06)

Francesca A. Lisi
University of Bari
Italy

Editor: Enrico Pontelli

The 2006 edition of the *Italian Conference on Computational Logic (CILC'06)* took place at the Department of Computer Science of the University of Bari, Italy, on June 26-27, 2006. It was the *21st* meeting of the Italian Association for Logic Programming (GULP) and affiliated to ALP.

The event was organized by the staff of the *Laboratory for Knowledge Acquisition and Machine Learning* (LACAM, Dept. of Computer Science, University of Bari) under the supervision of Floriana Esposito, Donato Malerba, and Giovanni Semeraro. The LACAM group is very active in the field of Inductive Logic Programming, both from the theoretical and the application side. I am myself a member of this group and I had the honor of starting the very first session of the convention with a tutorial ("Learning Rules on top of Ontologies: An Inductive Logic Programming Approach").

The program of CILC'06 was very stimulating. Most of the contributions were in the area of Knowledge Representation and Reasoning as one may expect considering the new challenges of the Semantic Web. A minor number of contributions came from the research areas on Agents, Learning, Constraints and Verification. The program committee selected 17 papers for presentation. The program was completed by an invited talk by Francesco M. Donini ("Knowledge Representation Tools for Electronic Commerce") also related to the Semantic Web. All the contributions and related slides of presentation are available on-line at the URL

http://cilc2006.di.uniba.it/programma.html

thanks to the webmaster Bruna Di Nanna.

Special thanks go to the local organizers, Pasquale Lops and Marco Degemmis, also members of the LACAM group. They managed to create a nice atmosphere especially during the lovely coffee and lunch breaks that were arranged inside the Department so that the participants could enjoy the air conditioning all along their stay at the convention. Who knows the Southern Italian summer can appreciate this special treatment!

2006 Doctoral Consortium in (Constraint) Logic Programming

Enrico Pontelli
New Mexico State University
USA

Editor: Enrico Pontelli

The 2006 ICLP Doctoral Consortium on (Constraint) Logic Programming took place on August 21st, 2006, during the 2006 International Conference on Logic Programming, in Seattle, WA.

The aims of the ICLP Doctoral Consortium are:

- To provide doctoral students working in the field of logic and constraint logic programming with a friendly and open forum to present their research ideas, listen to ongoing work from peer students, and receive constructive feedback
- To provide students with relevant information about important issues for doctoral candidates and future academics
- To develop a supportive community of scholars and a spirit of collaborative research.
- To support a new generation of researchers with information and advice on academic, research, industrial, and nontraditional career paths.

The Doctoral Consortium is designed for students currently enrolled in a Ph.D. program and conductive research in areas related to Logic or Constraint Logic Programming. The Consortium allows participants to interact with established researchers and with other students, through presentations, question-answer sessions, panel discussions, and invited presentations.

Six excellent submissions were accepted for the 2006 Doctoral Consortium. Each accepted student had the opportunity to present his/her research during a special session of ICLP 2006, under the supervision of an assigned mentor. The accepted submissions are:

- Martin Brain (University of Bath, UK): *"Declarative Problem Solving using Answer Set Programming"*
- Gergerly Lukacsy (Budapest Institute of Technology, Hungary): *"Description Logic Reasoning in Prolog"*
- Quan Phan (Catholic University of Leuven, Belgium): *"Static Memory Management for Logic Programming Languages"*
- Tu Huy Phan (New Mexico State University, USA): *"Efficient Reasoning about Action and Change in the Presence of Incomplete Information and its Application in Planning"*
- Tiago Soares (University of Porto, Portugal): *"Deductive Databases: Implementation, Parallelism, and Application"*
- Ka-Shu Wong (National ICT, Australia): *"Deducing Logic Programs"*

In addition to the students presentations and discussion, the 2006 Doctoral Consortium featured an invited presentation by Gopal Gupta, titled "Logic Programming: the grand unifying theory of Computer Science".

The Doctoral Consortium was made possible by generous contributions from

- the Association for Logic Programming
- the National Science Foundation
- the FLoC Federated Logic Conference organization
- the College of Arts & Sciences, New Mexico State University

that provided for financial support for the participating students.

2006 ICLP Doctoral Consortium: Gallery

The DC 2006 Chair :-)

Quan Phan (DC Participant)

Michel Ferreira (DC Mentor)

Inna Pivkina (DC Mentor)

David Scott Warren (DC Mentor)

Tiago Soares (DC Participant)

Son Cao Tran (DC Mentor)

Ka-Shu Wong (DC Participant)

Gergerly Lukacsy (DC Participant)

Martin Brain (DC Participant)

Agostino Dovier (DC Mentor)

Tu Huy Phan (DC Participant)

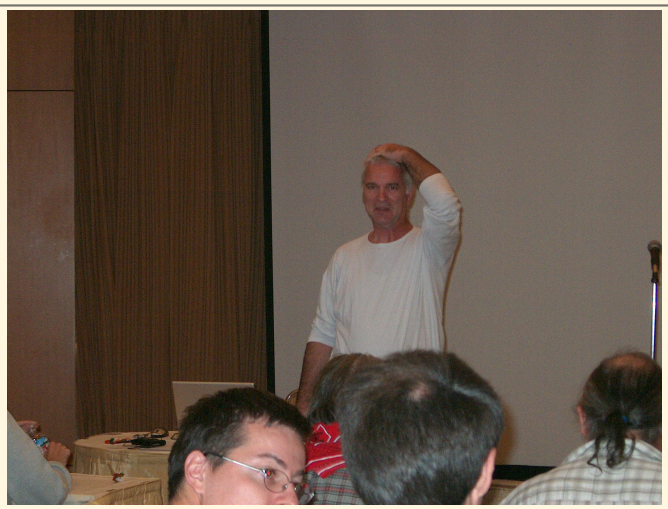
Gopal Gupta (Invited Speaker)

The 2006 DC Participants

International Conference on Logic Programming 2006: Gallery

Enrico Pontelli
New Mexico State University
USA

Editor: Enrico Pontelli



Bart providing instructions for the Programming Competition

The Prolog Programming Contest

The Prolog Programming Contest

ICLP 2006 Banquet

Andy King's Presentation

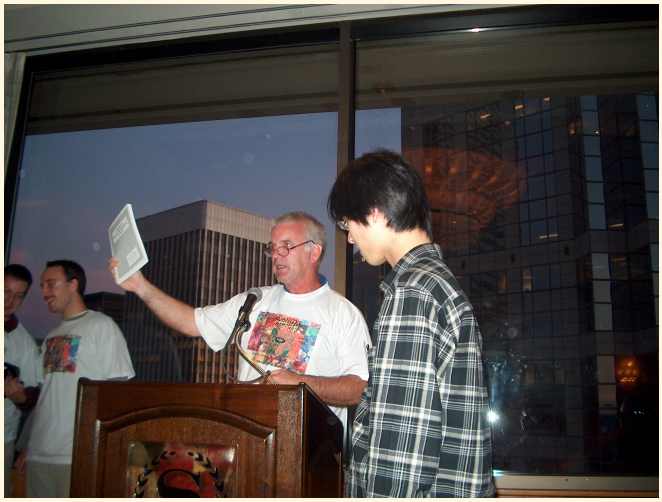
Sandro Etalle, ICLP 2006 Program Co-Chair

Manuel Hermenegildo, ALP President

Paper Award Ceremony

ICLP 2006 Banquet

ICLP 2006 Banquet



Prolog Programming Contest Winners

G. Gupta and P. Hill at the Awards Ceremony

S. Etalle and M. Truszczynski, ICLP 2006 Program Chairs

B. Demoen presenting the Programming Competition

Prolog Programming Contest Winners

ICLP Participants

Sunset on ICLP 2006

Program Transformation and OLDT a Personal Perspective

Taisuke Sato
Tokyo Institute of Technology
Japan

Editor: Enrico Pontelli

Warning: this article is compiled from my unreliable memory of the early days of program transformation in LP.

Long Long Ago

When I joined the Electrotechnical Laboratory (ETL), a research institute under the ministry of trade and industry in Japan more than thirty years ago, symbolic AI was on the rise. There was a strong group lead by Kazuhiro Fuchi, who initiated and orchestrated the Japanese Fifth Generation Computer System project (FGCS) starting in 1982, doing a variety of AI research from image understanding, speech recognition, natural language understanding, and so on.

What was lucky to me was that next year Koichi Furukawa, who became a deputy director of FGCS, brought Marseille Prolog into ETL after a visit to the Stanford Research Institute. I could have a chance of directly touching Prolog. Programs looked mysterious (predicate names were French!) and the interpreter written in FORTRAN ran very slowly but this new language seemed to promise an exciting future for me.

So I gradually shifted from Lisp that was the defacto standard AI language those days to Prolog that was born in Europe a couple years ago. In the mean time my research interest switched from natural language processing to Prolog. There were so many fun things to do from theory to application. In particular NAF (negation-as-failure) attracted me because firstly it was theoretically mysterious and secondly I simply believed that solving problems with NAF would lead to full first-order logic programming, the unexplored realm of logic programming. My problem was that I didn't have a concrete idea

about negation.

An Encounter with Partial Evaluation

In spring 1982, I went to Kyoto. My purpose was to attend the RIMS Symposium on Software Science and Engineering and to give a talk on intelligent backtracking in Prolog. In retrospect, it was a sheer coincidence that Yoshihiko Futamurna working at Hitachi also attended the symposium. He gave a talk on partial evaluation and he was a tireless advocator of partial evaluation though I did not know anything about him.

He explained so called "Futamura projection" to us and demonstrated how to construct a compiler from an interpreter, or more ambitiously a complier-compiler by partial evaluation. It was my first exposure to unfold/fold transformation and it was an exciting experience: it looked as if a theory, partial evaluation, was going to achieve a real break through, the construction of mathematically correct compiler. Although he talked in the context of a functional programming, obviously his
approach is applicable to Prolog. More to the point is that Prolog has unification that can eliminate all the difficulties in implementing a partial evaluator by say
Lisp.

Being galvanized by the possibility of applying partial evaluation to logic programs, I returned to ETL in Tsukuba. I tried a couple of transformation examples manually following Futamura's talk.The first one, I still remember, was to derive a recursive program for `initial(X,Z) - X` is an initial list of `Z` - from
`initial(X,Z):-append(X,Y,Z).` It was done easily. Other examples (reverse, a string matcher, etc) were also easy. I was quickly convinced of the power of unfold/fold transformation.

A few days later, encouraged by this initial success, I consulted Hisao Tamaki on this emerging research subfield, unfold/fold transformation in logic programming. He was staying at ETL as a visiting scholar then and had a strong background in programming language and algorithm. He immediately showed strong interest and it was a natural consequence that we began working in close collaboration beyond going for drink together.

Meaning Preserving Transformation

After starting collaboration, we knew that logic program transformation was not no-man's land. Quite opposite. There was already a bunch of work. Burstall and Darlington published their famous paper on functional program transformation in 1977. Keith Clark and Sharon Sickel applied their work to logic program derivation in the same year. Christopher Hogger published a rather comprehensive paper on deductive derivation of logic programs in J.ACM in 1981.

We noticed however that their framework is axiomatic deduction, and hence while every computed goal by a transformed program is a logical consequence of the (if-and-only-if completion of) original program, the converse does not hold in general. In other words, the equivalence of transformed programs in the sense of the least model semantics was not established yet.

Next year, 1983, was a busy year. While publishing two papers on LP, one in New Generation Computing 1(1) and the other at ICALP in Barcelona, we continued to try to prove equivalence among transformed programs. The point was clear: if `p(X):- q(X)` is immediately folded by `p(X):-q(X)`, we get `p(X):-p(X)`, which must be avoided. We needed some condition that prevents this folding.

Tamaki eventually hit the necessary idea. He introduced "rank" of a ground atom A based on the size of the smallest proof tree of A and applied it to determine a foldable condition and thus completed an equivalence proof. In the end he submitted a paper on the equivalence proof to the 2nd ICLP in Uppsala and I submitted a paper on program synthesis using unfold/fold transformation and negation to the 1st FGCS conference in Tokyo.

Append Optimizer

The mechanism of unfold/fold transformation is general but real problems are specific. To fill the gap is equivalent to applying it to a concrete problem. After publishing the unfold/fold paper, we looked for the next target.

Tamaki again hit a smart idea, "Append Optimizer". The story goes like this. First we allow a user to write a logic program freely using append/3. But append/3 incurs redundancy (copy). So we remove it by the Append Optimizer which automatically introduces new data structure similar to difference list to eliminate data copy by append/3.

It has turned out however that the Append Optimizer requires the dependency analysis of variables in a program. Worse yet an analyzer for this purpose often goes into infinite call as it traces the execution of the source program without any input (goals are completely general). Now the problem is to stop this infinite call. The idea of tabling came out this moment. Indeed necessity is the mother of invention: tabling can stop infinite call.

Putting aside the Append Optimizer, he concentrated on formalizing tabling in logic programming. He finally proved the completeness of OLDT (OLD with Tabulation) and presented it at the 3rd ICLP in London in summer 1986 when I stayed at Imperial College as a visiting researcher from ETL.

Looking Back

Regrettably the Append Optimizer project was never resumed after its suspension. I do not remember the exact reason. Probably because I was more involved in extending unfold/fold transformation to non-definite clauses and he interest turned to concurrency.

Our collaboration continued until 1989 when we published two papers on logic program synthesis. Later Hisao Tamaki left LP and moved to Toronto to study algorithm and I slowly began moving toward non-deductive approaches in AI such as GA and machine learning.

Looking back, even had we not proposed, probably someone would have introduced unfold/fold transformation and tabling to LP anyway, but we are happy that we could help their introduction to LP, which is all I can say now.

The Ciao Multiparadigm Language and Program Development Environment

The Ciao Development Team

Ciao is a modern, multiparadigm programming language with an advanced programming environment. It has a dual nature: on one hand it provides a high-performance, industrial quality, freely available, *ISO-standard-compliant* Prolog system. At the same time, its modular approach allows both *restricting* and *augmenting* the language through *libraries* in a well-controlled fashion. This allows providing significant extensions which make Ciao a truly *next-generation logic-programming language* as well as a *multiparadigm programming system.*

One of fundamental aspects of the Ciao approach is based on the observation that a single set of basic, well-chosen features (a *language kernel*) can effectively support several programming paradigms and styles [12,10,11]. This approach is, of course, not exclusive to Ciao, but in Ciao these facilities are uniformly available (and their use encouraged) from the system programmer level to the application programmer level.

The extensibility-based approach makes it possible to work with *fully declarative subsets* of logic programming and also to *extend* the core language both syntactically and semantically. Most importantly, these restrictions and extensions can be activated separately on a per-module basis without interfering with each other thanks to the notion of *packages* [4]. The different source-level constructs (and sub-languages / DSLs) are typically supported by compilation into the kernel language via *source-to-source* transformations, with the (rather infrequent) help of modules written in an external language using one of the several interfaces provided. Due to the existence of a common kernel, the programming styles that Ciao implements share much at both the semantic and implementation levels, and they naturally reuse significant portions of modern (C)LP implementation technology.

Ciao provides support for both *programming in the small* (by providing *scripts* and reduced-size executables, which include only those builtins and libraries used by the program) and *programming in the large.* Programming in the large is facilitated by its robust module system [4] and rich *assertion language* (another product of the "extensibility approach") combined with the capabilities of the Ciao preprocessor [2,13,14] for modular static verification, static debugging, and dynamic checking of such assertions.

Some of the principal distinguishing features of Ciao are:

ISO-Prolog: Ciao provides in its default mode an excellent Prolog system, without giving up on all of its "new-generation" features, thanks to the library-based approach. This distinguishes Ciao from other new-generation (C)LP systems that do not have an ISO-Prolog compliant mode.

Support for Multiple Programming Models and Paradigms: At the same time Ciao supports, *also via libraries*, different LP languages, several types of constraint domains (including CHR support), functional notation (including lazy evaluation), higher-order (with predicate/function abstractions), as well as several computation rules (Andorra model, breadth-first, iterative deepening, fuzzy Prolog) and object-oriented programming facilities. Modules written in ISO-Prolog can be combined freely with other modules written in these other supported paradigms, and some paradigms can even be mixed within the same module.

Powerful Preprocessor: A number of program analyzers, integrated in the Ciao preprocessor, allow inferring and checking many useful program properties. Most analyses are performed at the kernel language level, so that the same analyzers can be used for several of the supported programming models. The availability of this information provides quite unique functionality [14]:

- **Assertions and Program Debugging/Validation:** The properties that are statically inferred by the analyzers can be compared against programmer-provided (typically partial) specifications written using Ciao's unique *assertion language*. These properties include types/modes (data structure shape –including variable sharing– and instantiation state of variables), bounds on data sizes, determinacy, termination, non-failure, or bounds on resource consumption (time, space, or user-defined resources). The preprocessor *verifies* whether those properties are met by the actual code and *statically debugs* the program otherwise. Assertions that the system cannot prove nor disprove at compile-time can be optionally subject to *run-time checking*, with tests automatically added to the program. Both static and dynamic checking are safe in the sense that all errors flagged are violations of the specifications.
- **Assertions are not Compulsory:** A fundamental advantage of the Ciao approach in this context is that assertions *are not compulsory*. This distinguishes Ciao from other new-generation (C)LP systems where, e.g., type definitions or declarations are compulsory, and is of course instrumental in allowing Ciao to support Prolog.
- **Mobile Code Safety through Abstraction-Carrying Code:** When programs are validated the preprocessor can generate automatically certificates that can be attached to programs to be checked at the receiving end in order to guarantee compliance with a given safety policy [1].
- **Source-to-source Optimizations:** The information inferred by the global analyzers can be used to perform source-level code optimizations, including multiple abstract specialization, partial evaluation, dead code removal, goal reordering, parallelization with granularity control, reduction of concurrency / dynamic scheduling, low-level optimization, etc.

Rich Program Development Environment: In addition to all the facilities provided by the preprocessor, compiler, and top level, the programming environment includes:

- **Rich Graphical Development Interface:** based on the latest, graphical versions of Emacs (offering menu and widget-based interfaces with direct access to the top-level/debugger, preprocessor, and autodocumenter) as well as an embeddable source-level debugger with breakpoints, and several execution visualization tools. The environment provides also automated access to the documentation, extensive syntax highlighting, auto-completion, auto-location of errors in the source, etc., and is highly customizable. A plugin with very similar functionality is also available for the Eclipse environment.
- **Automatic Documentation Generation:** The assertions and directives present in the program and libraries, as well as all other program information available to the compiler, are used to generate automatically program documentation (including types/modes and other properties, machine-readable comments, etc.) by means of the Ciao autodocumenter [9].

Versatile, Incremental Compiler and Abstract Machine: A central piece of the system is the Ciao compiler, `ciaoc` [5] and its target abstract machine, which offer:

- **Modular, Incremental Compilation:** `ciaoc` performs automatically an incremental compilation which takes module dependencies into account without the need for *Makefiles*. Program modules can be linked statically, dynamically, or be automatically loaded on demand.
  The executables generated are competitive in both performance and size with current commercial and academic Prolog systems.
- **Versatile, Multiplatform Compilation Options:** Several types of executables can be built easily. In addition to the traditional Prolog top-level, the system offers support for the use of Ciao as a scripting language, for compilation to multiarchitecture *bytecode* executables, and for compilation to single-architecture, standalone executables. Multiple platforms are supported, including Windows, Linux, Mac Os X, and many other Un*x-based OSs. Optimizing compilation to native code (via C) is in a mature state [8] and will be part of the standard distribution in the near future.
- **Kernel Abstract Machine:** A comparatively simple, but optimized abstract machine supports the kernel language. It includes native support for *attributed variables* and for threading primitives, and a synchronization-enabled shared database [7].
- **Rich foreign interfaces:** Bidirectional foreign interfaces to C (with automatic generation of glue code), Java, TclTk, SQL databases (with a notion of predicate persistence), etc.

Support for Concurrency, Parallelism, and Distributed Execution: Ciao includes concurrency, parallelism, and distributed execution capabilities [3]. The notion of "active module" (or active object) allows compiling modules in such a way that they are ultimately mapped to a standalone process, which is transparently accessed by the rest of the application. In addition, the system also offers a full-fledged library for developing WWW-based applications [6].

Free Availability: Ciao is free software protected to remain so by the GNU LGPL license. It can be used freely to develop both free and commercial applications.

Finally, the system includes a **large set of libraries**, many of them contributed by users.

Probing Further

A recent motivational discussion of the fundamental design choices made in Ciao, as well as our thoughts about next generation programming systems can be found in [11] (`http://cliplab.org/papers/ciao-philosophy-note-tr.pdf`).

The reader is encouraged to explore the system, its documentation, and the tutorial papers that have been published on it, but, specially, to use it, of course! We are currently working on the new 1.14 system version which includes significant enhancements with respect to the previous version (1.10), including integration of the preprocessor and autodocumenter into the Ciao development tree as a single package (previously they had to be downloaded and installed separately). This version is available already on demand from the Ciao subversion repository.

Contact / download info:	The Ciao Development Team
`http://www.ciaohome.org`	Technical U. of Madrid, Spain
`http://www.cliplab.org`	U. of New Mexico, USA
`ciao@clip.dia.fi.upm.es`	U. Complutense de Madrid, Spain

Acknowledgments: The Ciao system is in continuous and very active development through the collaborative effort of numerous members of several institutions, including UPM, UNM, UCM, Roskilde U., U. Melbourne, Monash U., U. Arizona, Linköping U., NMSU, K. U. Leuven, Bristol U., Ben-Gurion U, INRIA, as well as many others. The development of the Ciao system has been supported by a number of European, Spanish, and other international projects (currently by EU IST-15905 *MOBIUS* project, the Spanish TIN-2005-09207 *MERIT* project, and the CAM *PROMESAS* program. Manuel Hermenegildo is also supported by the IST Prince of Asturias Chair at the University of New Mexico. The system documentation and related publications contain more specific credits for the many contributors to the system.

References

1. E. Albert, G. Puebla, and M. Hermenegildo. Abstraction-Carrying Code. In *Proc. of LPAR'04*, number 3452 in LNAI, pages 380–397. Springer-Verlag, 2005.
2. F. Bueno, P. Deransart, W. Drabent, G. Ferrand, M. Hermenegildo, J. Maluszynski, and G. Puebla. On the Role of Semantic Approximations in Validation and Diagnosis of Constraint Logic Programs. In *Proc. of the 3rd. Int'l Workshop on Automated Debugging–AADEBUG'97*, pages 155–170, Linköping, Sweden, May 1997. U. of Linköping Press.

3. D. Cabeza and M. Hermenegildo. Distributed Concurrent Constraint Execution in the CIAO System. In *Proc. of the 1995 COMPULOG-NET Workshop on Parallelism and Implementation Technologies*, Utrecht, NL, September 1995. U. Utrecht / T.U. Madrid. Available from `http://www.cliplab.org/`.
4. D. Cabeza and M. Hermenegildo. A New Module System for Prolog. In *International Conference on Computational Logic, CL2000*, number 1861 in LNAI, pages 131–148. Springer-Verlag, July 2000.
5. D. Cabeza and M. Hermenegildo. The Ciao Modular, Standalone Compiler and Its Generic Program Processing Library. In *Special Issue on Parallelism and Implementation of (C)LP Systems*, volume 30(3) of *Electronic Notes in Theoretical Computer Science*. Elsevier - North Holland, March 2000.
6. D. Cabeza and M. Hermenegildo. Distributed WWW Programming using (Ciao-)Prolog and the PiLLoW Library. *Theory and Practice of Logic Programming*, 1(3):251–282, May 2001.
7. M. Carro and M. Hermenegildo. Concurrency in Prolog Using Threads and a Shared Database. In *1999 International Conference on Logic Programming*, pages 320–334. MIT Press, Cambridge, MA, USA, November 1999.
8. M. Carro, J. Morales, H.L. Muller, G. Puebla, and M. Hermenegildo. High-Level Languages for Small Devices: A Case Study. In Krisztian Flautner and Taewhan Kim, editors, *Compilers, Architecture, and Synthesis for Embedded Systems*, pages 271–281. ACM Press / Sheridan, October 2006.
9. M. Hermenegildo. A Documentation Generator for (C)LP Systems. In *International Conference on Computational Logic, CL2000*, number 1861 in LNAI, pages 1345–1361. Springer-Verlag, July 2000.
10. M. Hermenegildo, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, P. López-García, and G. Puebla. The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems. In *Parallelism and Implementation of Logic and Constraint Logic Programming*, pages 65–85. Nova Science, Commack, NY, USA, April 1999.
11. M. Hermenegildo and The Ciao Development Team. Why Ciao? –An Overview of the Ciao System's Design Philosophy. Technical Report CLIP7/2006.0, Technical University of Madrid (UPM), School of Computer Science, UPM, December 2006. Available from: `http://cliplab.org/papers/ciao-philosophy-note-tr.pdf`.
12. M. Hermenegildo and The CLIP Group. Some Methodological Issues in the Design of CIAO - A Generic, Parallel, Concurrent Constraint System. In *Principles and Practice of Constraint Programming*, number 874 in LNCS, pages 123–133. Springer-Verlag, May 1994.
13. M. Hermenegildo, G. Puebla, and F. Bueno. Using Global Analysis, Partial Specifications, and an Extensible Assertion Language for Program Validation and Debugging. In K. R. Apt, V. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm: a 25–Year Perspective*, pages 161–192. Springer-Verlag, July 1999.
14. M. Hermenegildo, G. Puebla, F. Bueno, and P. López-García. Integrated Program Debugging, Verification, and Optimization Using Abstract Interpretation (and The Ciao System Preprocessor). *Science of Computer Programming*, 58(1–2):115–140, October 2005.

Most of these and other papers and technical reports related to Ciao can be obtained from the Clip lab main WWW server, `http://www.cliplab.org/`.

ALP Executive Committee Report

Editor: Maria Garcia de la Banda

The ALP Executive Committee had its annual meeting in Seattle (thanks this time to ICLP'06 and FLoC for getting us all together) with the usual long agenda which included issues such as the possible creation of a special interest group on computational logic; funding status; TPLP status; newsletter report; doctoral consortium report, future conference's organisational issues, and the ALP website.

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

- While the ALP EC indicated general support for the idea of creating Special Interest Group on computational logic (Siglog) in the ACM, the support had some reservations (including what would exactly mean for the ALP to be an associated association with the ACM). The decision was, therefore, to support the exploration of the possibility of creating an ACM Siglog.
- ICLP'07 will be held in Porto, with Veronica Dahl and Ilkka Niemela as co PC chairs. Possible co-locations and PC chairs for future conferences were discussed with several proposals received from, among others, Australia and Italy. It was agreed to publish a formal call for '08.
- There has been some concern regarding the lack of up to date information in the ALP website on TPLP. It was agreed that either procedures should be developed to maintain the information on TPLP activity up to date, or that pointer from the ALP web site should point to the CUP TPLP home page.
- While it was agreed that the Wikipedia entry for Prolog is not at all representative of the current state of logic programming, there was acknowledge the significant amount of work involved in updating it. It was thus suggested that those interested might look into trying to improve it.
- Given the difficulties found in increasing the participation of PhD students in the doctoral consortium and the the Newsletter's PhD thesis column, it was also agreed to improve their advertising by including it in the newsletter summary and drawing advisors' attention to it.
- ICLP 2006 has been quite successful; at the time of the meeting, there were

105 registered participants (35 of which were students). There was also a good response to the request for donations to ALP (through a voluntary $50 increased registration rate); 16 people contributed to this initiative.

Dissertations in Logic Programming

Contents

Deductive Multi-valued Model Checking

Ajay Mallya
University of Texas at Dallas

Model checking is widely used in the verification of computer systems. Classical model checking methods are based on the boolean interpretation of temporal logic formulas over Kripke structures. However, this method does not scale well and model checking of large systems is often intractable, due to the problem of state space explosion. Multi-valued logics go beyond the boolean interpretation of truth, by offering a multitude of truth values that can precisely encode contextual information, and at the same time provide reasonable generalizations of the operators of boolean logic.
In this dissertation, we develop the theory of multi-valued model checking, which is the multi-valued interpretation of formulas of temporal logic over multi-valued structures. The mechanism of Horn logic is used to specify the denotational semantics of temporal formulas over multi-valued Kripke structures, thereby yielding a systematic and efficient model checking procedure.

Multi-valued logics are a canonical form of the lattice-based theory of abstract interpretation. A complete abstraction procedure for Kripke structures is developed, using Belnap's four-valued logic as the abstract domain. For an infinite state system, that satisfies a given μ-calculus formula, a finite abstraction is produced, that also satisfies the same formula. The abstraction method is also extended to game theoretic generalizations of Kripke structures, to yield similar results.

Finally, the theory of co-Logic programming is used to develop model checking techniques for the verification of liveness properties using coinductive reachability analysis. Another application of co-Logic programming -- reasoning about

sequences of actions in real-time is also presented.

Logic Programming with Sets, Multisets, and Aggregates

Islam Elkabani
New Mexico State University

Logic Programming has began in the early 1970's as a direct outgrowth of earlier work in automatic theorem proving and artificial intelligence. The ideal of logic programming is purely *declarative programming*. This means that it brings to the programmer the benefit of only having to specify the logic component of an algorithm and leaving the control to the logic programming system.

Logic programming is based on *deductive reasoning* and this means that deriving conclusions based solely on what one already knows. This sort of reasoning is *monotonic*: as the set of beliefs grows, so does the set of conclusions that can be drawn from those beliefs. In recent years we witnessed an alternative logical systems, called *non-monotonic logics*, which allow new beliefs to retract existing conclusions. In particular, in the last few years a novel programming paradigm based on non-monotonic logics, has arisen, called Answer Sets Programming (ASP). ASP offers novel and declarative solutions in well-defined application areas, such as intelligent agents, planning, and diagnosis.

Nevertheless, there are simple properties, commonly encountered in real-world applications, that cannot be conveniently handled within the current framework of ASP - such as properties dealing with arithmetic and aggregation. In particular, aggregates and other forms of set and multiset constructions have been shown to be essential to reduce the complexity of software development and to improve the declarative level of the programming framework.

In this dissertation an extension of ASP which supports a semantically well-founded, flexible, and efficient implementation of aggregates has been developed. Also a generic framework which provides a simple and elegant treatment of extensions of ASP w.r.t. generic constraint domains has been offered and an instantiation of this framework to the case of constraint theory for aggregates is developed. Finally, a new logic programming language that supports both extensional and intensional multiset constraints is presented. The broader impact of this dissertation is to efficiently extend ASP with aggregates and multiset constraints, leading to a more compact ASP programs, allow as many problems as possible to be described in an intuitive and more declarative

manner in ASP and reduce the level of complexity in solving ASP programs.

Community News

List of News:

Special Issues

Systems Announcements

Fellowships, Awards, Job Announcements

Other Announcements

Constraints Journal: Call for Papers
Special Issue on Abstraction and Automation in Constraint Modeling

Communicated by Alan Frisch

GUEST EDITORS

Alan Frisch, University of York, U.K.
Ian Miguel, University of St. Andrews, U.K.

INTRODUCTION

Constraint Programming (CP) is a powerful technology that has been successfully used for tackling a wide range of real-life, complex applications. To solve a problem with CP it first needs to be modelled by a set of constraints that must be satisfied by any solution.
Because formulating such a model, and especially formulating one that is solvable in practice, is often difficult, CP technology is currently accessible to only a small number of experts. For CP to be more widely used by non-experts, more research effort is needed in order to ease the use of CP technology.

One way of improving usability is by extending CP technology to enable models to be formulated at a higher level of abstraction. For instance, support for set variables (variables whose domain values are sets) in many constraint languages and solvers has abstracted away from the low-level details of how the set variable is represented; the user no longer needs to know these details. However, variables that take certain other types of values, such as functions and relations, are not yet supported directly by constraint solvers. In this case, the abstract variable can be refined into a representation that comprises a set of more primitive variables and a collection of constraints among them. In order to avoid forcing the user to perform this step manually, automated refinement is a key goal.

Automation can also aid the modelling process by transforming a constraint model into one that can be solved more effectively. Such transformations include adding implied constraints, adding symmetry-breaking constraints, adding constraints to exploit dominances in optimisation problems, removing propagation-redundant constraints and creating relaxed versions of the initial problem.

This special issue is devoted to the development and use of abstraction and automation facilities in constraint modelling.
We invite submissions from interested authors in this challenging and important area.

Topics of Interest include, but are not limited to:

- The use of abstraction facilities in formulating models.
- Abstraction in constraint languages.
- Abstract variable domains.
- Abstraction in search control.
- Automated refinement.
- Automated generation of implied constraints.
- Automated generation of symmetry-breaking constraints.
- Automated generation of constraints to exploit dominances.
- Automated generation of relaxations.

Paper Submission

Researchers are invited to submit original papers that make a significant contribution to the field to ianm@cs.st-and.ac.uk. (Note that the usual on-line submission procedure for the Constraints journal will not be followed initially for the Special Issue). All submissions should be in .pdf format and follow Constraints Journal guidelines. Papers of at most 30 journal pages are preferred.

When submitting, please use the subject "Constraints Special Issue Paper Submission" and clearly specify the e-mail address and phone number of the corresponding author. Receipt of papers will be acknowledged. Submissions will be reviewed by at least two reviewers. All accepted papers will meet the usual high-quality standards of the Constraints Journal.

Authors intending to submit should send an expression of interest (including a provisional title, list of authors and a few sentences outlining the topic of the paper) to ianm@cs.st-and.ac.uk by May 1st, 2007.

Important Dates

Expression of interest: May 1st, 2007
Submission of papers: July 1st, 2007
Notification of acceptance: October 1st, 2007 Final versions of accepted papers: Dec 1st, 2007.
Expected publication of the special Issue: 2nd issue of 2008 (Apr 1st).

Important Links

- Special issue home page: **http://www.cs.st-andrews.ac.uk/~ianm/ModellingSpecialIssue.html**
- Constraints journal home page: **http://ai.uwaterloo.ca/~vanbeek/Constraints/constraints.html**
- Guidelines for authors: **http://ai.uwaterloo.ca/~vanbeek/Constraints/Instructions_for_Authors.html**

Fellowship Announcement
Kurt Gödel Centenary Prize Fellowship

Communicated by Kurt Gödel Society

The Kurt Gödel Society is proud to announce the commencement of the research fellowship prize program in honor and celebration of Kurt Gödel's 100th birthday.
The research fellowship prize program sponsored by the John Templeton Foundation will offer: two Ph.D. (pre-doctoral) fellowships of $60,000 US per annum for two years two post-doctoral fellowships of $ 80,000 US per annum for two years one senior fellowship of $120,000 US per annum for one year

The purpose of the fellowship is to support original research in mathematical logic, “meta-mathematics,” philosophy of mathematical logic, and the foundations of mathematics. This fellowship is to carry forward the legacy of Gödel, whose works exemplify deep insights and breakthrough discoveries in mathematical logic.

The selection will be made based upon an open, international competition. An international Board of Jurors chaired by Professor Harvey Friedman will oversee the process. The finalist papers will be published in a special issue of a premier journal in mathematical logic.

Web:**http://kgs.logic.at/goedel-fellowship**
Contact: goedel-fellowship@logic.at

Goal and Criteria of Merit:

In pursuit of similar insights and discoveries, we adopt the following criteria of merit for evaluating Fellowship applications:
1. Intellectual merit, scientific rigor and originality of the submitted paper and work plan. The paper should combine visionary thinking with academic excellence.
2. Potential for significant contribution to basic foundational understanding of logic and the likelihood for opening new, fruitful lines of inquiry.

3. Impact of the grant on the project and likelihood that the grant will make this new line of research possible.
4. The probability that the pursuit of this line of research is realistic and feasible for the applicant.
5. Qualifications of the applicants evaluated via CV and recommendation letters* (*recommendation letters are not required for senior applications).

Scopes:

Original fellowship proposals from all fields of mathematical logic (such as Computability Theory, Model Theory, Proof Theory, Set Theory), meta-mathematics, the philosophy of mathematics, and the foundations of mathematics insofar as the research has strong relevance or resemblance to the Gödelian insights and originality.

Preliminary Timeline

December 1, 2006. Announcement
June 15, 2007. Submissions deadline
October 2007. Jury decision due on papers to be published
December 15, 2007. Final versions due
January 2008. Jury decision on winners due
February 2008. Award Ceremony
Mar.-Sept.2008. Commencement of the Fellowships

Journal on Satisfiability, Boolean Modeling, and Computation: Call for Papers Application of Constraints to Formal Verification

Communicated by Miroslav Velev

We would like to invite you to submit a paper to the special issue of the Journal on Satisfiability, Boolean Modeling and Computation (JSAT) on the topic of application of constraints to formal verification (CFV).

The submission deadline is January 10, 2007.

Topics include, but are not limited to, the following:

- application of constraint solvers to hardware verification;
- application of constraint solvers to software verification;
- dedicated solvers for formal verification problems;
- tuning SAT for formal verification and testing;
- challenging formal verification problems.

The submissions have to be in the JSAT format: **http://www.isa.ewi.tudelft.nl/Jsat/** and have to be e-mailed to: **mvelev@gmail.com**
If possible, please confirm your intent to submit a paper.

We look forward to your submission,

Miroslav Velev and Joao Marques-Silva
Editors of the special issue of JSAT on CFV

Software Announcement Bedwyr: A Proof Search Approach to Model Checking

Communicated by Alwen Tiu

Dear Colleagues,
we would like to announce the first release of Bedwyr, an extended logic programming language that allows model checking directly on syntactic expression possibly containing bindings.

Bedwyr allows simple and declarative executable specifications of operational semantics of various programming languages with bindings, type systems and process calculi, and also supports automatic reasoning for some simple properties about the specifications. The current distribution of Bedwyr contains a collection of examples involving pi-calculus, including:

- different styles of specifications of operational semantics: late transition system, transition system with abstraction/concretion, polyadic pi calculus, and spi-calculus,
- (open, weak) bisimulation,
- modal logics,
- (symbolic) trace analyses: for checking correspondence assertions, finding attacks on protocols specified in spi-calculus, and finding

"separating traces" of processes.

Bedwyr makes use of a simple form of tabling to support proof search for inductive and co-inductive specifications (e.g., bisimulation checking for non-terminating processes).

You will find a general description of Bedwyr below this message. More details can be found on Bedwyr website **http://slimmer.gforge.inria.fr/bedwyr/**

Sincerely,
Bedwyr developers

Bedwyr
A proof search approach to model checking
http://slimmer.gforge.inria.fr/bedwyr/

Bedwyr is a programming framework written in OCaml that facilitates natural and perspicuous presentations of rule oriented computations over syntactic expressions and that is capable of model checking based reasoning over such descriptions.
Bedwyr is in spirit a generalization of logic programming. However, it embodies two important recent observations about proof search:

1. It is possible to formalize both finite success and finite failure in a sequent calculus; proof search in such a proof system can capture both may and must behavior in operational semantics specifications.
2. Higher-order abstract syntax can be supported at a logical level by using term-level lambda-binders, the nabla-quantifier, higher-order pattern unification, and explicit substitutions; these features allow reasoning directly on expressions containing bound variables.

The distribution of Bedwyr includes illustrative applications to the finite pi-calculus (operational semantics, bisimulation, trace analyses and modal logics), the spi-calculus (operational semantics), value-passing CCS, the lambda-calculus, winning strategies for games, and various other model checking problems. These examples should also show the ease with which a new rule based system and particular questions about its properties can be be programmed in Bedwyr. Because of this characteristic, we believe that the system can be of use to people interested in the broad area of reasoning about computer systems.

The present distribution of Bedwyr has been developed by the following individuals:

David Baelde & Dale Miller (INRIA & LIX/Ecole Polytechnique)
Andrew Gacek & Gopalan Nadathur (University of Minneapolis)
Alwen Tiu (Australian National University and NICTA).

In the spirit of an open source project, we welcome contributions in the form of example applications and also new features from others.

Book Announcement
Handbook of Modal Logic

Communicated by Wiebe va der Hoek

Dear reader,

We are very pleased to inform you that the above book has now been published. We hope that you will be as pleased with the final result as we are.

The book is also announced on Elsevier's website. For more information, please go to:

http://books.elsevier.com/uk//elsevier/uk/subindex.asp?maintarget=&isbn=0444516905&country=United+Kingdom&srccode=&ref=CWS1&subcode=&head=&pdf=&basiccode=&txtSearch=&SearchField=&operator=&order=&community=elsevier

Wiebe van der Hoek

PostDoc and Ph.D. Positions
Action Languages and Answer Set Programming

Communicated by Torsten Schaub

Dear colleague,

the research unit for Systems Biology GoFORSYS offers interdisciplinary PostDoc and Ph.D. students positions, among others, in the area of Action Languages and Answer Set Programming for Modeling Biological Networks within the KRR group at the University of Potsdam.

The official job advertisement can be found at **http://www.goforsys.org/positions.php**

For further information about these position please contact:

Prof. Torsten Schaub
Fon +49-331-977-3080/3081
Fax +49-331-977-3122
Email torsten@cs.uni-potsdam.de

Please send your application to the address indicated on the website.

Best regards, -torsten schaub

Ph.D. Positions
Symbolic Computation at RISC-Linz

Communicated by Guenter Landsmann

Symbolic computation is the branch of mathematics and computer science which solves problems on symbolic objects representable on a computer.

RISC-Linz, the Research Institute for Symbolic Computation of the Johannes Kepler University in Linz, Austria, is one of the world's leading institutions for research and education in this new and promising area. RISC-Linz invites students for its well established Ph.D. program in symbolic computation. For excellent applicants we offer fellowships covering tuition and living expenses.

Applications for studies starting in October should be received by February 15.

For a description of application details and the RISC curriculum, see the web page **http://www.risc.uni-linz.ac.at/education/phd/**

The RISC Ph.D. Coordinator
phd-coordinator@risc.uni-linz.ac.at

Software Announcement
QBF Certificates: "ozziKs" Now Available

Communicated by Marco Benedetti

When a quantified CSP (QCSP) is claimed to be TRUE by some solver, how can we verify that such answer is correct?

In a standard CSP, it is just a matter of checking (in polynomial time) the values assigned to the variables by the solver against each constraint in the CSP.

In the QCSP framework things are more complex: As a certificate we need a strategy. A strategy can be seen as a set of functions - one for each existentially quantified variable - yielding admissible values for existential variables as a function of some relevant subset of the universally quantified ones.

One such strategy is what we can actually check against the set of constraints to certify its validity.

Also, a strategy is a piece of information which is itself interesting (provided a "real-world" problem has been encoded into the underlying QCSP) as it provides an explicit scenario in which the validity of the quantified set of constraints is revealed.

For the case of quantified boolean constraints (QBFs), which we address here, strategies are also called simply certificates.

The software "ozziKs", whose public availability we publicise here, is the implementation of an algorithm that:builds a certificate of satisfiability C(F) - a.k.a. strategy or policy or quantified model - for any given true Quantified Boolean Formula F for which a suited inference log is available (at present, only the QBF solver sKizzo produces a suited log); verifies C(F) against F, thus certifying in a solver-independent way the validity of F; evaluates user-provided expressions of various kinds over C(F); writes to file in different formats C(F) and/or the result of the evaluation of the above mentioned expressions.

A detailed user manual is available here. Linux, OS-X, and cygwin versions are available for download at **http://sKizzo.info.**

Marco Benedetti, PhD
Research Associate
LIFO - University of Orleans
BP 6759 - 45067 Orleans (France)
Tel: +33 (0)2 38 49 47 96
Fax: +33 (0)2 38 41 71 37
E-mail: **Marco.Benedetti@univ-orleans.fr**

Information & Computation: Call for Papers
Structural Operational Semantics

Communicated by Rob van Glabbeek

AIM:

Structural operational semantics (SOS) provides a framework for giving operational semantics to programming and specification languages. A growing number of programming languages from commercial and academic spheres have been given usable semantic descriptions by means of structural operational semantics. Because of its intuitive appeal and flexibility, structural operational semantics has found considerable application in the study of the semantics of concurrent processes. Moreover, it is becoming a viable alternative to denotational semantics in the static analysis of programs, and in proving compiler correctness.

Recently, structural operational semantics has been successfully applied as a formal tool to establish results that hold for classes of process description languages. This has allowed for the generalisation of well-known results in the field of process algebra, and for the development of a meta-theory for process calculi based on the realization that many of the results in this field only depend upon general semantic properties of language constructs.

This special issue aims at documenting state-of-the-art research, new developments and directions for future investigation in the field of structural operational semantics. Specific topics of interest include (but are not limited to):

- programming languages
- process algebras
- higher-order formalisms
- rule formats for operational specifications
- meaning of operational specifications
- comparisons between denotational, axiomatic and operational semantics
- compositionality of modal logics with respect to operational specifications
- congruence with respect to behavioural equivalences
- conservative extensions
- derivation of proof rules from operational specifications
- software tools that automate, or are based on, SOS.

Papers reporting on applications of SOS to software engineering and other areas of computer science are welcome.

This special issue is an outgrowth of the series of SOS workshops, which started in 2004, and serves in part as a opportunity to publish the full versions of the best papers presented at SOS 2006. However, papers that were not presented at SOS 2006 are equally welcome, and all submissions will be refereed and subjected to the same quality criteria, meeting the standards of Information and Computation.

Papers submitted to the special issue must contain original material that has not previously been published, and parallel submission for publication elsewhere is not allowed. However, an extended abstract or short version of the paper may be submitted for presentation at the SOS 2007 workshop, which will take place before the publication of the special issue.

PAPER SUBMISSION:

We solicit unpublished papers reporting on original research on the general theme of SOS. Papers should take the form of a dvi, postscript or pdf file. We recommend following Elsevier's instructions at

http://authors.elsevier.com/JournalDetail.html?PubID=622844

and using LaTeX2e with documentclass elsart.

IMPORTANT DATES:

- Submission of tentative title and abstract: 15 December 2006
- Submission of full paper: 15 February 2006

CONTACT and submission address:

sos2006@cs.stanford.edu

EDITORS of this special issue:

Rob van Glabbeek
National ICT Australia
Locked Bag 6016
University of New South Wales
Sydney, NSW 1466
Australia

Peter D. Mosses
Department of Computer Science
Swansea University
Singleton Park
Swansea SA2 8PP
United Kingdom

Software Announcement
Web Interfaces for the ECCE and LOGEN Specializers

Communicated by Michael Leuschel

We would like to announce the availability of web interfaces for two logic program specialization tools: *Ecce* and *Logen*.

Ecce is an automatic online program specialiser for pure Prolog programs (with built-ins). It takes a pure Prolog program and a query of interest and then specialises the program for that particular query.

LOGEN is an offline partial evaluation system for (almost) full Prolog written using the so called "cogen approach". Basically, the cogen is a system which:

1. based upon an annotated version of the program to be specialised produces a specialised partial evaluator for that program. This partial evaluator is called a generating extension
2. the generating extension can be used to specialise the program in a very efficient manner.

The web interfaces for both of these tools are available via:

http://www.stups.uni-duesseldorf.de/software.php

The web interfaces allow one to use those systems without installation on a local machine. They also provide an intuitive access to the various command-line options. In the case of Logen a graphical way to annotate the source programs is provided. Both web interfaces were developed within the EU-funded project ASAP (a web interface to all of ASAP tools, containing analysis and slicing tools can be found at **http://www.stups.uni-duesseldorf.de/~asap/asap-online-demo/asap.php)**.

A brief overview of these tools and their web interfaces can be found here:

http://www.stups.uni-duesseldorf.de/publications_detail.php?id=140

The choice on which tool to use depends on the particular application.
Ecce is a fully automatic online specialiser. It is hence easier to use by inexperienced users, but more difficult to tweak in case specialisation does not proceed as desired. Ecce uses more refined control techniques and can perform conjunctive partial deduction and slicing, which Logen cannot. Because of the additional annotation phase, Logen is more difficult to master by inexperienced users, but the outcome of the specialisation is easier to tweak and predict. The specialisation phase of Logen is very efficient, with very little overhead compared to ordinary evaluation. Logen can deal with almost full Prolog, whereas Ecce only deals with declarative Prolog programs. The latter restricts the range of programs to which Ecce can be applied, but it also allows Ecce to perform more powerful optimisations.

Book Announcement
Lectures on the Curry-Howard Isomorphism

Communicated by M.H. Sorensen & P. Urzyczyn

Lectures on the Curry-Howard Isomorphism
Studies in Logic and the Foundations of Mathematics, 149
by Morten Heine Sorensen, Pawel Urzyczyn
ISBN: 0-444-52077-5, 456 pages

This is an entirely rewritten book version of the DIKU lecture notes published online in 1998. The book gives an intruduction to various topics related to the formulas-as-types analogy, in particular:

- Type-free and simply-typed lambda-calculus
- Intuitionistic logic
- Combinatory logic
- Classical logic and control operators
- Sequent calculus
- Dialogue games
- Intuitionistic arithmetic and Godel's system T
- Second-order logic and polymorphism
- Dependent types and pure type systems.

The book contains a large number of exercises, many of these accompanied with extensive hints and solutions.

PostDoc Position
IRISA & INRIA-Rennes

Communicated by Jean-Pierre Talpin

The Espresso team at INRIA-Rennes (Brittany, France) is seeking for a post-doctorate to work on the modular verification and validation of GALS software architectures in avionics.

The Espresso team (**http://www.irisa.fr/espresso**) develops Polychrony, an embedded software design tool based on a synchronous multi-clocked model of computation.

The topic of the post-doctorate is the verification globally asynchronous and locally synchronous software architectures within the open-source design workbench Topcased (**http://www.topcased.org**).
More specifically, the post-doctorate program is interested in the design and implementation of verification techniques, using synchronous and asynchronous model-checkers, for the modular verification and validation of globally asynchronous and locally synchronous software architectures.

The performance of model-checkers used and tools designed in the frame of the program will be assessed by case studies such as a flight guidance system. The post-doctorate will carry out the definition and implementation of verification functionalities in the forthcomming Eclipse plugin of the Polychrony tool, developped by the team.
Preference will be given to candidates with a background, doctorate study and research interests in the area of formal methods, verification and model-checking, and interest in model-driven engineering for embedded systems.

The position provides an opportunity to join a large research institute and engage in research on formal methods for embedded software engineering in collaboration with an important academic and industrial consortium.
The selected post-doctorate candidate will be appointed as expert engineer for an initial and renewable period of 18 monthes. As expert engineer, the post-doctorate will receive a competitive monthly salary of 2750=80 including social coverages.

Applications, including a vitae, references and a brief description of research interests should be sent in reply of the present e-mail.

Awards Announcement
Automated Reasoning Challenge

Communicated by Geoff Sutcliffe

The MPTP $100 Challenges are sets of classical first-order reasoning problems, expressed in the TPTP language, to be proved by fully automated reasoning systems, within specified reasonable resource constraints. The challenge problems are based on the Mizar Mathematical Library. The goal of the MPTP challenges is to boost the development of automated theorem proving and artificial intelligence methods and tools for reasoning in large theories that involve large numbers of consistently used concepts. Details of the challenges, including instructions for entering, are available at ...

http://www.tptp.org/MPTPChallenge/

The winners of the MPTP challenges will be announced at CADE-21, and will each receive $100 in real US dollars ... who says there's no money in ATP! Results on the challenge problems, for publicly available systems, are on the web page.
If your browser is up to it, the interactive interface is fun to use :-)

Cheers,

Josef (Urban) and Geoff

Software Announcement
ECLiPSe now Open Source and under MPL

Communicated by Joachim Schimpf

The ECLiPSe Constraint Logic Programming System has recently been open-sourced by its current owner Cisco Systems, and is available under the terms of an MPL (Mozilla Public Licence) equivalent.

For those not so familiar with licensing terms, this means essentially:

- the system can now be used by anyone for any purpose, there is no longer a restriction to academic use
- any modification to the source code must be made available under the same licence, i.e. contributed back
- added libraries and application code are not affected by the licence and can, for instance, remain proprietary

We hope that this step will encourage the wider use of ECLiPSe in both the academic and commercial world. The new setup will give us better

flexibility to work with contributors from the user base, integrating more of the interesting work that is being done with ECLiPSe, and making it available to a wider community. Cisco itself is going to continue contributing to maintenance and development, and so is the other current commercial user, CrossCore Optimization.

The first open-source release is labelled ECLiPSe 5.10 and can be downloaded from either **http://www.eclipse-clp.org** or from **http://www.sourceforge.net/projects/eclipse-clp**
The official source repository now also resides on the Sourceforge site.

Enjoy!

The ECLiPSe Team

Constraint: Call for Papers
Constraint Based Methods for Bioinformatics

Communicated by Alessandro Dal Palù

INTRODUCTION

Bioinformatics is a challenging and fast growing area of research, which is of utmost importance for our understanding of life. Major contributions to this discipline can provide significant benefits in medicine, agriculture, and industry. To pick out only a few examples, Bioinformatics tackles problems related to:

- Recognition, analysis, and organization of DNA sequences
- Biological systems simulations (for metabolic or regulatory networks)
- Prediction of the spatial conformations and interactions of biological polymers (e.g., proteins, RNA)

Recently, these problems have been formalized and studied using constraints (often over finite domains or intervals of reals). Biology is a source of extremely interesting and computationally expensive tasks, that can be encoded exploiting the application of recent and more general techniques of constraint programming.

As evidence of this trend, various workshops (Constraints and Bioinformatics/Biocomputing at CP97, CP98 and Constraint based methods for Bioinformatics at ICLP2005 and CP2006) witnessed the interest and the importance of research in the topic and moreover presented new developments of constraint technology (see, e.g., details in **http://www.dimi.uniud.it/dovier/WCB06**).

We believe that is valuable to gather papers addressing the application of constraints to biological problems. With this special issue, we desire to reflect the state of the art of this field, and thus we seek for papers that report new ideas, advances and results. The submission of papers is opened to anyone who developed ideas and/or obtained results in the bioinformatics area making use of constraint programming.

PAPER SUBMISSION

Researchers are invited to submit original or survey papers to all the editors (addresses below). The editors would appreciate receiving a tentative short abstract (may be partial) at the beginning of January. In the first email, please specify the title, keywords, abstract and the author's email addresses. Receipt of submissions will be acknowledged. All final submissions should be in .pdf format, and must adhere to the Constraints Journal guidelines **http://ai.uwaterloo.ca/~vanbeek/Constraints/Instructions_for_Authors.html** (Note that the usual on-line submission procedure for the Constraints Journal will not be followed initially for the Special Issue)

We expect papers no longer than 25 pages, but this is not a strict constraint. Submissions will be reviewed by at least two reviewers. All accepted papers will meet the usual high-quality standards of the Constraints Journal.

IMPORTANT DATES

Abstract Submission:	January 10th, 2007
Submission Deadline:	January 31st, 2007
Notification of Acceptance:	April 30th, 2007
Final Version of Accepted Papers:	June 30th, 2007

GUEST EDITORS

Alessandro Dal Palu', Parma University, Italy.
Email alessandro.dalpalu AT unipr.it

Agostino Dovier, Udine University, Italy.
Email dovier AT dimi.uniud.it

Sebastian Will, Freiburg University, Germany.
Email will AT informatik.uni-freiburg.de

See: **http://www.dimi.uniud.it/dovier/WCBSI/**

Logic-Programming Related Call for Papers

Contents

TABLEAUX 2007

Aix en Provence, France, July 3-6, 2007

http://tableaux2007.univ-cezanne.fr/

IMPORTANT DATES

Workshop proposal submission deadline: **December 5, 2006**
Notification of acceptance of workshops: **December 15, 2006**
Tutorial proposal submission deadline: **January 10, 2007**
Notification of acceptance of tutorials: **January 20, 2007**
Title and abstract submission deadline: **February 2, 2007**
Paper submission deadline: **February 9, 2007**
Notification of acceptance of papers: **April 2, 2007**
Final version of papers due: **April 16, 2007**
Conference: **July 3-6, 2007**

GENERAL INFORMATION

This conference is the 16th in a series of international meetings on Automated Reasoning with Analytic Tableaux and Related Methods.
In July 2007, the conference will be held in Aix en Provence, France. The conference proceedings will be published in LNAI series as in the previous editions of the conference.
See **http://tableaux2007.univ-cezanne.fr/** for more information on TABLEAUX 2007, and http://i12www.ira.uka.de/TABLEAUX for information about the TABLEAUX conference series.

TOPICS

Tableau methods are a convenient formalism for automating deduction in various non-standard logics as well as in classical logic. Areas of application include verification of software and computer systems, deductive databases, knowledge representation and its required inference engines, and system diagnosis. The conference brings together researchers interested in all aspects - theoretical foundations, implementation techniques, systems development and applications - of the mechanization of reasoning with tableaux and related

methods.
Topics of interest include (but are not restricted to):

- analytic tableaux for various logics (theory and applications)
- related techniques and concepts, e.g., model checking and BDDs
- related methods (model elimination, sequent calculi, connection method, ...)
- new calculi and methods for theorem proving in classical and non-classical logics (modal, description, intuitionistic, linear, temporal, many-valued...)
- systems, tools, implementations and applications.

TABLEAUX 2007 puts a special emphasis on applications. Papers describing applications of tableaux and related methods in areas such as, for example, hardware and software verification, knowledge engineering, semantic web, etc. are particularly invited.
One or more tutorials and workshops will be part of the conference program.

SUBMISSIONS

The conference will include contributed papers, tutorials, system descriptions, position papers and invited lectures. Submissions are invited in four categories:
A Research papers (reporting original theoretical and/or experimental research, up to 15 pages)
B System descriptions (up to 5 pages)
C Position papers and brief reports on work in progress
D Tutorials in all areas of analytic tableaux and related methods from academic research to applications (proposals up to 5 pages)

Submissions in categories A and B will be reviewed by peers, typically members of the program committee. They must be unpublished and not submitted for publication elsewhere. Accepted papers in these categories will be published in the conference proceedings . For category B submissions a working implementation must exist and be available to the referees.
Submissions in category C will be reviewed by members of the program committee and a collection of the accepted papers in this category will be published as a Technical Report of the LSIS/Université Paul Cézanne.
Tutorial submissions (Category D) may be at introductory, intermediate, or advanced levels. Novel topics and topics of broad interest are preferred. The submission should include the title, the author, the topic of the tutorial, its level, its relevance to conference topics, and a description of the interest and the scientific contents of the proposed tutorial. Tutorial proposals will be reviewed by members

of the program committee.

Authors of accepted papers are expected to present their work at the conference.

CALL FOR WORKSHOP PROPOSALS

TABLEAUX 2007 launches a Call for Workshop Proposal on specialised subjects in the range of the conference topics. We can accept up to 2 proposals. The proposals are reviewed by members of the PC committee. The purpose of a workshop is to offer an opportunity of presenting novel ideas, ongoing research, and to discuss the state of the art of an area in a less formal but more focused way than the conference itself. It is also a good opportunity for young researchers to present their own work and to obtain feedback. The format of a workshop is left to the organizers, but it is expected to contain significant time for discussion. The intended schedule is for one-day workshops.
Further information and instructions about submissions will be available on the conference website at **http://tableaux2007.univ-cezanne.fr/**.

IEEE Symposium on Logic in Computer Science
Wroclaw, Poland, July 10-14, 2007

http://www.informatik.hu-berlin.de/lics/lics07

Theme:

Suggested, but not exclusive, topics include:

- automata theory,
- automated deduction,
- categorical models and logics,
- concurrency and distributed computation,
- constraint programming,
- constructive mathematics,
- database theory,
- domain theory,
- finite model theory,
- formal aspects of program analysis,

- formal methods,
- hybrid systems,
- lambda and combinatory calculi,
- linear logic,
- logical aspects of computational complexity,
- logics in artificial intelligence,
- logics of programs,
- logic programming,
- modal and temporal logics,
- model checking,
- probabilistic systems,
- process calculi,
- programming language semantics,
- reasoning about security,
- rewriting,
- specifications,
- type systems and type theory, and verification.

We welcome submissions in emergent areas, such as bioinformatics and quantum computation, if they have a substantial connection with logic.

All submissions must be electronic.

Deadlines:

Paper Registration and Abstract Submission: 15 January 2007
Paper Submission: 22 January 2007

Programme Committee:

- Albert Atserias (Technical U. Catalonia),
- Steve Awodey (Carnegie Mellon U.),
- Nachum Dershowitz (Tel Aviv U.),
- Thomas Ehrhard (Paris 7 and CNRS),
- Javier Esparza (U. of Stuttgart),
- Marcelo Fiore (U. of Cambridge),
- Erich Graedel (RWTH Aachen),
- Tom Henzinger (EPFL),
- Alan Jeffrey (Bell Labs),
- Achim Jung (U. of Birmingham),

- Dexter Kozen (Cornell U.),
- Kim Larsen (Aalborg U.),
- Jerzy Marcinkowski (U. of Wroclaw),
- Luke Ong (Chair, U. of Oxford),
- Frank Pfenning (Carnegie Mellon U.),
- Andrew Pitts (U. of Cambridge),
- Vladimiro Sassone (U. of Southampton),
- Nicole Schweikardt (Humboldt U. Berlin),
- Peter Selinger (Dalhousie U.),
- Natarajan Shankar (CS Lab. SRI),
- Victor Vianu (UC San Diego),
- Igor Walukiewicz (CNRS and Bordeaux)

Workshop on Computing with Terms and Graphs

Braga, Portugal, March 31, 2007

http://www.termgraph.org.uk/2007

The advantage of computing with graphs rather than terms is that common subexpressions can be shared, improving the efficiency of computations in space and time. Sharing is ubiquitous in implementations of programming languages: many functional, logic, object-oriented and concurrent calculi are implemented using term graphs. Research in term and graph rewriting ranges from theoretical questions to practical implementation issues.

Topics include: the modelling of first- and higher-order term rewriting by (acyclic or cyclic) graph rewriting, the use of graphical frameworks such as interaction nets and sharing graphs (optimal reduction), rewrite calculi for the semantics and analysis of functional programs, graph reduction implementations of programming languages, graphical calculi modelling concurrent and mobile computations, object-oriented systems, graphs as a model of biological or chemical abstract machines, and automated reasoning and symbolic computation systems working on shared structures.

All submissions must be done electronically. Please email your submission to **mackie@lix.polytechnique.fr**

Submission Deadline : December 29 2006.

Program committee.

- Zena Ariola (University of Oregon, USA),
- Andrea Corradini (University of Pisa, Italy),
- Maribel Fernandez (King's College London, UK),
- Bernhard Gramlich (Vienna University of Technology, Austria),
- Annegret Habel (University of Oldenburg, Germany),
- Claude Kirchner (LORIA, France),
- Jean-Jacques Levy (INRIA, France),
- Ian Mackie (King's College London and Ecole Polytechnique, France (Co-Chair)),
- Aart Middeldorp (University of Innsbruck, Austria),
- Ugo Montanari (University of Pisa, Italy),
- Jorge Sousa Pinto (University of Minho, Braga, Portugal),
- Detlef Plump (University of York, UK (Co-Chair))
- Arend Rensink (University of Twente, NL)

Conference on Rewriting Techniques and Applications

Paris, France, June 26-28, 2007

http://www.lsv.ens-cachan.fr/rdp07/rta.html

THEME.

RTA is the major forum for the presentation of research on all aspects of rewriting. Typical areas of interest include (but are not limited to):

- Applications: case studies; rule-based (functional and logic) programming; symbolic and algebraic computation; theorem proving; system synthesis and verification; analysis of cryptographic protocols; proof checking; reasoning about programming languages and logics; program transformation;

- Foundations: matching and unification; narrowing; completion techniques; strategies; constraint solving; explicit substitutions; tree automata; termination; combination;
- Frameworks: string, term, graph, and proof rewriting; lambda-calculus and higher-order rewriting; proof nets; constrained rewriting/ deduction; categorical and infinitary rewriting; integration of decision procedures;
- Implementation: compilation techniques; parallel execution; rewrite tools; termination checking;
- Semantics: equational logic; rewriting logic; rewriting models of programs.

SUBMISSIONS.

Submission categories include regular research papers and system descriptions. Problem sets and submissions describing interesting applications of rewriting techniques are also welcome. The page limit for submissions is 15 pages in Springer LNCS style (10 pages for system descriptions). The submission Web page will be made available beginning of December. As usual, the proceedings of RTA'07 will be published in the Springer LNCS series.

IMPORTANT DATES:

Jan 26, 2007: Deadline for electronic submission of title and abstract
Jan 31, 2007: Deadline for electronic submission of papers
Apr 02, 2007: Notification of acceptance of papers
Apr 23, 2007: Deadline for final versions of accepted papers

CONFERENCE CHAIRS:

- Ralf Treinen (Cachan, France),
- Xavier Urbain (Paris, France)

Typed Lambda Calculi and Applications

Paris, France, July 26-28, 2007

http://www.lsv.ens-cachan.fr/rdp07/tlca.html

The TLCA series of conferences serves as a forum for presenting original research results that are broadly relevant to the theory and applications of typed calculi.
The following list of topics is non-exhaustive: Proof-theory: Natural deduction and sequent calculi, cut elimination and normalisation, linear logic and proof nets, type-theoretic aspects of computational complexity / Semantics: Denotational semantics, game semantics, realisability, categorical models / Implementation: Abstract machines, parallel execution, optimal reduction, type systems for program optimisation/ Types: Subtypes, dependent types, type inference, polymorphism, types in theorem proving / Programming: Foundational aspects of functional and object-oriented programming, proof search and logic programming, connections between and combinations of functional and logic programming, type checking.

The programme of TLCA will consist of three invited talks and about 25 papers selected from original contributions. Accepted papers will be published as a volume of Springer Lecture Notes in Computer Science series (**http://www.springer.de/comp/lncs/index.html**).

Important Dates:
December 22 Title and abstract due
January 2 Deadline for submission
March 10-15 Author review period
March 25 Notification of acceptance-rejection
April 20 Deadline for the final version

Program Committee:

- Chantal Berline (CNRS),
- Peter Dybjer (Chalmers),
- Healfdene Goguen (Google),
- Robert Harper (Carnegie Mellon U.),
- Olivier Laurent (CNRS),
- Simone Martini (U. of Bologna),
- Simona Ronchi Della Rocca (Chair, U. of Torino),
- Peter Selinger (U. of Dalhousie),
- Paula Severi (U. of Leicester),
- Kazushige Terui (U. of Sokendai),
- Pawel Urzyczyn (U. of Warsaw)

Workshop on Logic, Language, Information and Computation
Rio de Janeiro, Brasil, July 2-5, 2007

http://www.cin.ufpe.br/~wollic/wollic2007

WoLLIC is an annual international forum on inter-disciplinary research involving formal logic, computing and programming theory, and natural language and reasoning. Each meeting includes invited talks and tutorials as well as contributed papers.

PAPER SUBMISSION

Contributions are invited on all pertinent subjects, with particular interest in cross-disciplinary topics. Typical but not exclusive areas of interest are: foundations of computing and programming; novel computation models and paradigms; broad notions of proof and belief; formal methods in software and hardware development; logical approach to natural language and reasoning; logics of programs, actions and resources; foundational aspects of information organization, search, flow, sharing, and protection. Proposed contributions should be in English, and consist of a scholarly exposition accessible to the non-specialist, including motivation, background, and comparison with related works. They must not exceed 10 pages (in font 10 or higher), with up to 5 additional pages for references and technical appendices. The paper's main results must not be published or submitted for publication in refereed venues, including journals and other scientific meetings.
It is expected that each accepted paper be presented at the meeting by one of its authors.

Papers must be submitted electronically at **www.cin.ufpe.br/~wollic/wollic2007/instructions.html**

A title and single-paragraph abstract should be submitted by February 23, and the full paper by March 2 (firm date). Notifications are expected by April 13, and final papers for the proceedings will be due by April 27 (firm date).

STUDENT GRANTS

ASL sponsorship of WoLLIC'2007 will permit ASL student members to apply for a modest travel grant (deadline: April 1, 2007). See **www.aslonline.org/studenttravelawards.html** for details.

IMPORTANT DATES

February 23, 2007: Paper title and abstract deadline
March 2, 2007: Full paper deadline (firm)
April 12, 2007: Author notification
April 26, 2007: Final version deadline (firm)

International Conference on Logic Programming
Porto, Portugal, September 8-13, 2007

http://www.dcc.fc.up.pt/iclp07

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 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.
- Language Issues: Concurrency, Objects, Coordination, Mobility, Higher Order, Types, Modes, Programming Techniques.
- Alternative Paradigms: Abductive Logic Programming, Answer Set Programming, Constraint Logic Programming, Inductive Logic Programming, Alternative Inference Engines and Mechanisms.
- Applications: Deductive Databases, Data Integration, Software Engineering, Natural Language, Web Tools, Internet Agents, Artificial Intelligence, Bioinformatics.

The three broad categories for submissions are:

1. technical papers, where specific attention will be given to work providing novel integrations of the areas listed above,
2. application papers, where the emphasis will be on their impact on the application domain as opposed to the advancement of the the state-of-the-art of logic programming, and
3. posters, ideal for presenting and discussing current work not yet ready for publication, for PhD thesis summaries and research project overviews.

In addition to papers and posters, the technical program will include invited talks, tutorials, a Doctoral Consortium, and workshops.

Papers and Posters

Papers and posters must describe original, previously unpublished research, and must not simultaneously be submitted for publication elsewhere. They must be written in English. Technical papers and application papers must not exceed 15 pages in the Springer LNCS format (cf. http://www.springer.de/lncs/). The limit for posters is 2 pages in that format.

Publication

It is expected that the proceedings will be published by Springer-Verlag in the LNCS series. All accepted papers and abstracts of accepted posters will be included in the proceedings.

Important Dates

Paper registration deadline: March 2, 2007
Submission deadline: March 9, 2007
Notification of authors: May 4, 2007
Camera-ready copy due: June 8, 2007

ICLP 2007 Organization

Program Co-chairs:
Verónica Dahl and Ilkka Niemelä
General Chair:

Fernando Silva

Local chair:

Ricardo Rocha

Publicity Chair:

Salvador Abreu

Contact Address: iclp07@dcc.fc.up.pt

Program Committee:

Maurice Bruynooghe
Keith Clark
Verónica Dahl (Co-chair)
Marina De Vos
Yannis Dimopoulos
Inês Dutra
Esra Erdem
Maurizio Gabbrielli
Patricia M Hill
Katsumi Inoue
Tomi Janhunen
Tony Kusalik
Nicola Leone
Vladimir Lifschitz
Ilkka Niemelä (Co-chair)
Luís Moniz Pereira
German Puebla
Francesca Rossi
Kostis Sagonas
Peter Schachte
Torsten Schaub
Guillermo R. Simari
Tran Cao Son
Paul Tarau
Francesca Toni
Eric Villemonte de la Clergerie
David S. Warren
Stefan Woltran

Conference Venue

ICLP 2007 will be held in the city of Porto, second largest in Portugal. Porto is located by the Douro river and the Atlantic, has a truly unique appearance with many striking bridges, a historic center classified by UNESCO as a World Heritage site, a new House of Music by Rem Koolhaas and a nice Museum of Modern Art (Museu de Serralves). Porto is also well known for the much celebrated Port wine grown in the Douro valley. The conference will feature a cruise in the Douro river along with other optional tours.

Trends in Functional Programming
New York, NY, April 2-4, 2007

http://tltc.shu.edu/tfp2007/

The symposium on Trends in Functional Programming (TFP) is an international forum for researchers with interests in all aspects of functional programming languages, focusing on providing a broad view of current and future trends in Functional Programming. It aspires to be a lively environment for presenting the latest research results through acceptance by extended abstracts. A formal post-symposium refereeing process then selects the best articles presented at the symposium for publication in a high-profile volume.

TFP 2007 is co-hosted by Seton Hall University and The City College of New York (CCNY) and will be held in New York, USA, April 2-4, 2007 at the CCNY campus.

The TFP symposium is the successor to the successful series of Scottish Functional Programming Workshops. Previous TFP symposia were held in Edinburgh, Scotland in 2003 (co-located with IFL), in Munich, Germany in 2004, in Tallinn, Estonia in 2005 (co-located with ICFP and GPCE), and in Nottingham, UK in 2006 (co-located with Types). For further general information about TFP please see the TFP homepage at http://cs.shu.edu/tfp2007/ .

SCOPE OF THE SYMPOSIUM

The symposium recognizes that new trends may arise through various routes. As part of the Symposium's focus on trends we therefore identify the following five article categories. High-quality articles are solicited in any of these categories:

Research Articles leading-edge, previously unpublished research work

Position Articles	on what new trends should or should not be
Project Articles	descriptions of recently started new projects
Evaluation Articles	what lessons can be drawn from a finished project
Overview Articles	summarizing work with respect to a trendy subject

Articles must be original and not submitted for simultaneous publication to any other forum. They may consider any aspect of functional programming: theoretical, implementation-oriented, or more experience-oriented. Applications of functional programming techniques to other languages are also within the scope of the symposium.

Articles on the following subject areas are particularly welcomed:

- Dependently Typed Functional Programming
- Validation and Verification of Functional Programs
- Debugging for Functional Languages
- Functional Programming and Security
- Functional Programming and Mobility
- Functional Programming to Animate/Prototype/Implement Systems from Formal or Semi-Formal Specifications
- Functional Languages for Telecommunications Applications
- Functional Languages for Embedded Systems
- Functional Programming Applied to Global Computing
- Functional GRIDs
- Functional Programming Ideas in Imperative or Object-Oriented Settings (and the converse)
- Interoperability with Imperative Programming Languages
- Novel Memory Management Techniques
- Parallel/Concurrent Functional Languages
- Program Transformation Techniques
- Empirical Performance Studies
- Abstract/Virtual Machines and Compilers for Functional Languages
- New Implementation Strategies
- any new emerging trend in the functional programming area

If you are in doubt on whether your article is within the scope of TFP, please contact the TFP 2007 program chair, Marco T. Morazan, at tfp2007@shu.edu.

BEST STUDENT PAPER AWARD

TFP traditionally pays special attention to research students, acknowledging that

students are almost by definition part of new subject trends. A prize for the best student paper is awarded each year.

SUBMISSION AND DRAFT PROCEEDINGS

Acceptance of articles for presentation at the symposium is based on the review of extended abstracts (6 to 10 pages in length) by the program committee. Accepted abstracts are to be completed to full papers before the symposium for publication in the draft proceedings and on-line.
The submission must clearly indicate to which category it belongs to: research, position, project, evaluation, or overview paper. It should also indicate whether the main author or authors are research students. Formatting details can be found at the TFP 2007 website. Submission procedures will be posted on the TFP 2007 website as the submission deadline is reached.

The papers in the draft proceedings will also be made available on-line under the following conditions, with which all authors are asked to agree:

> *The documents distributed by this server have been provided by the contributing authors as a means to ensure timely dissemination of scholarly and technical work on a noncommercial basis. Copyright and all rights therein are maintained by the authors or by other copyright holders, notwithstanding that they have offered their works here electronically. It is understood that all persons copying this information will adhere to the terms and constraints invoked by each author's copyright. These works may not be reposted without the explicit permission of the copyright holder.*

POST-SYMPOSIUM REFEREEING AND PUBLICATION

In addition to the draft symposium proceedings, we intend to continue the TFP tradition of publishing a high-quality subset of contributions in the Intellect series on Trends in Functional Programming. All TFP authors will be invited to submit revised papers after the symposium. These will be refereed using normal conference standards and a subset of the best papers, over all categories, will be selected for publication. Papers will be judged on their contribution to the research area with appropriate criteria applied to each category of paper.

Student papers will be given extra feedback by the Program Committee in order to assist those unfamiliar with the publication process.

IMPORTANT DATES

Abstract Submission: February 1, 2007
Notification of Acceptance: February 20, 2007
Registration Deadline: March 2, 2007
Camera Ready Full Paper Due: March 9, 2007
TFP Symposium: April 2-4, 2007

ORGANIZATION

Symposium Chair:	Henrik Nilsson, University of Nottingham, UK
Programme Chair:	Marco T. Morazan, Seton Hall University, USA
Treasurer:	Greg Michaelson, Heriot-Watt University, UK
Local Arrangements:	Marco T. Morazan, Seton Hall University, USA

Static Analysis Symposium

Kongens Lyngby, Denmark, August 22-24, 2007

http://www.imm.dtu.dk/sas2007

Static Analysis is increasingly recognized as a fundamental tool for high performance implementations and verification of programming languages and systems. The series of Static Analysis Symposia has served as the primary venue for presentation of theoretical, practical, and application advances in the area.

The technical programme for SAS 2007 will consist of invited lectures, tutorials, panels, presentations of refereed papers, and software demonstrations. Contributions are welcome on all aspects of Static Analysis, including, but not limited to:

- abstract domain
- abstract interpretation
- abstract testing
- compiler optimisations
- control flow analysis
- data flow analysis
- model checking
- program specialization

- security analysis
- theoretical analysis frameworks
- type based analysis
- verification systems

Submissions can address any programming paradigm, including concurrent, constraint, functional, imperative, logic and object-oriented programming. Survey papers, that present some aspect of the above topics from a new perspective, and application papers, that describe experience with industrial applications, are also welcome. Papers must describe original work, be written and presented in English, and must not substantially overlap with papers that have been published, or that are simultaneously submitted to a journal or a conference with refereed proceedings.

Submitted papers should be at most 15 pages formatted in LNCS style excluding bibliography and well-marked appendices not intended for publication). PC members are not required to read the appendices, and thus papers should be intelligible without them. The proceedings will be ublished by Springer-Verlag in the Lecture Notes in Computer Science series.

Program Committee

- Agostino Cortesi (U. Venice, Italy)
- Patrick Cousot (ENS, France)
- Manuel Fahndrich (Microsoft, USA)
- Gilberto Filé (U. Padova, Italy, co-chair)
- Roberto Giacobazzi (U. Verona, Italy)
- Chris Hankin (Imperial College, UK)
- Manuel Hermenegildo (TU. Madrid, Spain)
- Jens Knoop (TU. Vienna, Austria)
- Naoki Kobayashi (Tohoku U., Japan)
- Julia Lawall (U. Copenhagen, Denmark)
- Hanne Riis Nielson (DTU, Denmark, co-chair)
- Andreas Podelski (U. Freiburg, Germany)
- Jakob Rehof (U. Dortmund, Germany)
- Radu Rugina (Cornell U., USA)
- Mooly Sagiv (Tel-Aviv U., Israel)
- Dave Schmidt (Kansas State U., USA)
- Helmut Seidl (TUM, Germany)
- Harald Søndergaard (U. Melbourne, AU)

- Kwangkeun Yi (Seoul N. U., Korea)

Organising committee

- Christian W. Probst
- Flemming Nielson
- Terkel K. Tolstrup
- Henrik Pilegaard
- Eva Bing
- Elsebeth Strøm

Important dates

Submission of abstract:	March 26, 2007
Submission of full paper:	March 30, 2007
Notification:	May 7, 2007
Camera-ready version:	June 4, 2007
Conference:	August 22-24, 2007

Conference on Integration of AI and OR

Brussels, Belgium, May 23-26, 2007

http://www.cs.brown.edu/sites/cpaior07

After a successful series of five international workshops (Ferrara, Paderborn, Ashford, Le Croisic, and Montreal) and three international conferences (Nice, Prague, Cork), the fourth international conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming for Combinatorial Optimization Problems will be held in Brussels, Belgium, in 2007.

The aim of the conference is to bring together interested researchers from constraint programming (CP), artificial intelligence (AI) and operations research (OR) to present new techniques or new applications in combinatorial optimization and to provide an opportunity for researchers in one area to learn about techniques in the others. A main objective of this conference series is also to give these researchers the opportunity to show how the integration of techniques from different fields can lead to interesting results on large and complex problems.

Therefore papers that actively combine, integrate, or contrast approaches from more than one of the areas are especially solicited. High quality papers from a single area are also welcome. Finally, application papers showcasing CP/AI/OR techniques on innovative and challenging applications or experience reports on such applications are strongly encouraged.

CP-AI-OR'07 will be preceded by a Master Class where leading researchers give introductory and overview talks. This year, the topic of the Master Class will be on "Constraint-Based Scheduling". The Master Class is intended for PhD students, researchers, and practitioners.

Papers should be at most 15 pages in length, and should be prepared in the format used for the Springer Lecture Notes in Computer Science series (http://www.springer.de/comp/lncs/authors.html). The proceedings will be published in the Springer Lecture Notes in Computer Science series. All papers are to be submitted electronically in a PDF or PS format by following the instructions on the conference site.

Following the conference, authors of all accepted papers will be invited to submit substantially extended versions of their papers to a special issue of Constraints devoted to papers from CP-AI-OR'07. These papers will undergo an additional, very thorough refereeing process and a selection of the best papers will be published.

Important Dates

Submission: January 26, 2007
Notification: February 26, 2007
Camera-ready: March 7, 2007
Master Class: May 23rd, 2007

Computability in Europe
Siena, Italy, June 18-23, 2007

http://www.mat.unisi.it/newsito/cie07.html

The programme committee of CiE 2007 cordially invites all researchers (European

and non-European) in computability related areas to submit their papers (in PDF-format, max 10 pages) for presentation at CiE 2007: see the conference website (above) for the online submission procedure.

The conference proceedings will be published by Springer Lecture Notes in Computer Science (LNCS). There will also be journal special issues: APAL, JLC, TCS-C, ToCS - to which full versions of selected submissions to CiE 2007 will be invited to be submitted. For a list of conference topics see: http://www.amsta.leeds.ac.uk/~pmt6sbc/cie07.descr.html#themes

IMPORTANT DATES:

Submission of Papers: Jan. 12, 2007
Notification of Authors: Feb. 16, 2007
Deadline for Final Revisions: Mar. 9, 2007
Deadline for Submission of Informal Presentations: Apr. 27, 2007

PLENARY AND TUTORIAL SPEAKERS:

- PIETER ADRIAANS (Amsterdam) - Learning as Data Compression
- KOBI BENENSON (Harvard) - Biological Computing
- ANNE CONDON (Vancouver) - Computational challenges in prediction and design of nucleic acid structure
- STEPHEN COOK (Toronto) - Low Level Reverse Mathematics
- YURI ERSHOV (Novosibirsk) - tba
- WOLFGANG MAASS (Graz) - Theoretical Aspects of Biological Computation
- SOPHIE LAPLANTE (Paris) - Using Kolmogorov Complexity to Define Individual Security of Cryptographic Systems
- ANIL NERODE (Cornell) - Logic and Control
- ROGER PENROSE (Oxford) - tba
- MICHAEL RATHJEN (Leeds) - Theories and Ordinals in Proof Theory
- DANA SCOTT (Carnegie Mellon) - Two Categories for Computability
- ROBERT I. SOARE (Chicago) - Computability and Incomputability
- PHILIP WELCH (Bristol) - tba

SPECIAL SESSIONS SPEAKERS:

- Doing without Turing Machines: Constructivism and Formal Topology (Chairs: Giovanni Sambin, Dieter Spreen):
    - Andrej Bauer (Ljubljana)

- Douglas Bridges (Canterbury, NZ)
- Thierry Coquand (Goeteborg)
- Martin Escardo (Birmingham)
- Maria Emilia Maietti (Padua)

- Approaches to Computational Learning (Chairs: Marco Gori, Franco Montagna):
  - John Case (Newark, Delaware)
  - Klaus Meer (Odense)
  - Frank Stephan (Singapore)
  - Osamu Watanabe (Tokyo)

- Real Computation (Chairs: Vasco Brattka, Pietro Di Gianantonio):
  - Pieter Collins (Amsterdam)
  - Abbas Edalat (London)
  - Hajime Ishihara (Tokyo)
  - Robert Rettinger (Hagen)
  - Martin Ziegler (Paderborn)

- Computability and Mathematical Structure (Chairs: Serikzhan Badaev, Marat Arslanov):
  - Vasco Brattka (Cape Town)
  - Barbara F. Csima (Waterloo)
  - Sergey S. Goncharov (Novosibirsk)
  - Jiri Wiedermann (Prague)
  - Liang Yu (Nanjing)

- Complexity of Algorithms and Proofs (Chairs: Elvira Mayordomo, Jan Johannsen):
  - Eric Allender (Rutgers)
  - Joerg Flum (Freiburg)
  - Michal Koucky (Prague)
  - Neil Thapen (Prague)
  - Heribert Vollmer (Hannover)

- Logic and New Paradigms of Computability (Chairs: Paola Bonizzoni, Olivier Bournez):
  - Felix Costa (Lisbon)
  - Natasha Jonoska (Tampa, Florida)
  - Giancarlo Mauri (Milan)

  - Grzegorz Rozenberg (Leiden)
  - Damien Woods (Cork)

- Computational Foundations of Physics and Biology (Chairs: Guglielmo Tamburrini, Christopher Timpson):
  - James Ladyman (Bristol)
  - Itamar Pitowsky (Jerusalem)
  - Grzegorz Rozenberg (Leiden)
  - Giuseppe Trautteur (Naples)

WOMEN IN COMPUTABILITY WORKSHOP in association with the Computer Research Association's Committee on the Status of Women in Computing Research (CRA-W) Organisers: Paola Bonizzoni, Elvira Mayordomo Speakers: Anne Condon (Vancouver), Natasha Jonoska (Florida), Carmen Leccardi (Milan), and others

CiE 2007 will be co-located with CCA 2007, the annual CCA (Computability and Complexity in Analysis) Conference (Siena, College Santa Chiara, June 16-18, 2007): **http://cca-net.de/cca2007/**

Symposium on Frontiers of Combining Systems

Liverpool, U.K., September 10-12, 2007

http://www.csc.liv.ac.uk/~frocos07/

Topics

Typical topics of interest include (but are not limited to):

- combinations of logics such as combined predicate, temporal, modal, or epistemic logics;
- combinations and modularity in ontologies;
- combination of decision procedures, of satisfiability procedures, and of constraint solving techniques;
- combinations and modularity in term rewriting;
- integration of equational and other theories into deductive systems;
- combination of deduction systems and computer algebra;

- integration of data structures into CLP formalisms and deduction processes;
- hybrid methods for deduction, resolution and constraint propagation;
- hybrid systems in knowledge representation and natural language semantics;
- combined logics for distributed and multi-agent systems;
- logical aspects of combining and modularising programs and specifications.

Background

In various areas of computer science, such as logic, computation, program development and verification, artificial intelligence, and automated reasoning, there is an obvious need for using specialised formalisms and inference mechanisms for special tasks. In order to be usable in practice, these specialised systems must be combined with each other, and they must be integrated into general purpose systems. The development of general techniques and methods for the combination and integration of special formally defined systems, as well as for the analysis and modularisation of complex systems has been initiated in many areas. The International Symposium on Frontiers of Combining Systems (FroCoS) traditionally focuses on this type of research questions and activities and aims at promoting progress in the field. The previous FroCoS's were held in Munich (1996), Amsterdam (1998), Nancy (2000), Santa Margherita Ligure (2002), and Vienna (2005). In 2004 and 2006, FroCoS joined IJCAR, the International Joint Conference on Automated Reasoning. Like its predecessors, FroCoS 2007 wants to offer a common forum for research activities in the general area of combination, modularisation and integration of systems (with emphasis on logic-based ones), and of their practical use.

Proceedings

Proceedings will be published by Springer in the Lecture Notes on Artificial Intelligence (LNAI) series.

Other Events

FroCoS will be collocated with FTP (Workshop on First-order Theorem Proving)

Important Dates

April 23, 2007: Abstract submission deadline
April 30, 2007: Full paper submission deadline

June 5, 2007: Notification of acceptance
June 20, 2007: Camera ready copies due

Submission

The programme committee seeks high-quality submissions that are original and not submitted for publication elsewhere. There are two categories of submission:

1. Regular papers. Submissions should not exceed 15 pages and should contain original research, and sufficient detail to assess the merits and relevance of the contribution. Detailed instructions can be found at the FroCoS website.
2. Tool descriptions. Submissions should not exceed 8 pages, and should describe the implemented tool and its novel features. Detailed instructions can be found at the FroCoS website.

Programme Chair: Frank Wolter, Liverpool, UK
Conference Chair: Boris Konev, Liverpool, UK

Programme Committee:

- Alessandro Armando
- Franz Baader
- Jacques Calmet
- Silvio Ghilardi
- Bernhard Gramlich
- Deepak Kapur
- Boris Konev
- Till Mossakowski
- Joachim Niehren
- Albert Oliveras
- Dirk Pattinson
- Silvio Ranise
- Mark Reynolds
- Christophe Ringeissen
- Ulrike Sattler
- Amilcar Sernadas
- Cesare Tinelli
- Luca Vigano
- Frank Wolter

Workshop on Satisfiability Modulo Theories
Berlin, Germany, July 1-2, 2007

http://www.lsi.upc.edu/~oliveras/smt07

Background

Deciding the satisfiability of first-order formulas modulo background theories, known as the Satisfiability Modulo Theories (SMT) problem, has proved to be useful in verification, compiler optimization, scheduling, and other areas.

The success of SMT techniques depends on the development of both domain-specific decision procedures for each concrete theory (e.g. linear arithmetic, the theory of arrays, or the theory of bit-vectors) and combination methods that allow one to obtain more versatile SMT tools. These two ingredients together make SMT techniques well-suited for use in larger automated reasoning and formal verification efforts.

Aims and Scope

The aim of the workshop is to bring together researchers and users of SMT tools and techniques. Continuing with the PDPAR tradition, we especially encourage submission of papers focused on pragmatic aspects. Relevant topics include but are not limited to:

- New decision procedures and new theories of interest
- Combination of decision procedures
- Novel implementation techniques
- Benchmarks and evaluation methodologies
- Applications and case studies
- Theoretical results

Important dates

Submission deadline : 23 April
Notification of acceptance/rejection : 21 May
Final version due : 4 June

Workshop : 1-2 July

Proceedings

Given the informal style of the workshop, only informal proceedings will be distributed at the workshop. We are planning to publish a selected subset of the submitted papers as post-proceedings in a special volume of the Electronic Notes in Theoretical Computer Science (ENTCS) unless the authors prefer not to.

Paper Submission and Proceedings

Following the PDPAR'06 initiative, there are two categories of submissions:

- Original papers: contain original research (simultaneous submissions are not allowed) and sufficient detail to assess the merits and relevance of the submission. For papers reporting experimental results, authors are strongly encouraged to make their data available. Given the informal style of the workshop, work in progress will be welcome.
- Presentation-only papers: describe work recently published or submitted and will not be included in the proceedings. We see this as a way to provide additional access to important developments that SMT Workshop attendees may be unaware of.

Papers in both categories will be peer-reviewed. Papers should not exceed 10 pages (Postscript or PDF) and should be written in LaTeX, 11pt, one column, a4paper, standard margins. Technical details may be included in an appendix to be read at the reviewers' discretion. Full submission guidelines are at the workshop web page.

Program Chairs

Sava Krstic, Intel Corporation
Albert Oliveras, Tech. Univ. of Catalonia

Program Committee

- Clark Barrett, New York University
- Alessandro Cimatti, ITC-Irst, Trento
- Byron Cook, Microsoft Research
- Amit Goel, Intel Corporation

- Aarti Gupta, NEC Labs America
- Shuvendu Lahiri, Microsoft Research
- Leonardo de Moura, Microsoft Research
- Robert Nieuwenhuis, Technical University of Catalonia
- Silvio Ranise, LORIA, Nancy
- Roberto Sebastiani, Universita` di Trento
- Ofer Strichman, Technion
- Cesare Tinelli, University of Iowa
- Ashish Tiwari, Stanford Research Institute (SRI)

Workshop on Rule-based Programming

Paris, France, June 29, 2007

http://www.lsv.ens-cachan.fr/rdp07/rule.html

Scope

Rule-based programming provides a framework that facilitates viewing computation as a sequence of changes transforming a complex shared structure such as a term, graph, proof, or constraint store. In rule-based languages, a set of abstractions and primitive operations typically provide sophisticated mechanisms for recognizing and manipulating structures. In a classical setting, a rule-based program consists of a collection of (conditional) rewrite rules together with a partially-explicit specification of how the rule collection should be applied to a given structure.

Due to theoretical and technological advances, rule-based programming techniques are being incorporated into a wide range of research areas including: Generative
Programming, Aspect-Oriented Programming, Software Maintenance, Reverse Engineering, Domain Specific Language Development, and Information Assurance (e.g., security, testing, etc.). Oftentimes, the ad hoc incorporation of rule-based techniques into a particular area or problem domain raises general issues that warrant further study. Related to this is a growing need to share foundational infrastructure (e.g., parsers, pretty printers, etc.) between rule-based systems. The goal of this workshop is to
foster the exchange of ideas within the rule-based programming community.

Topics

We solicit original papers on all topics related to rule-based programming including:

- Applications to Software
  - Development
  - Evolution
  - Information Assurance
- Systems
  - Descriptions of rule-based systems
  - Descriptions of rule-based languages
- Hybrid paradigms -- Rule-based programming combined with:
  - Functional programming
  - Logic programming
  - OO programming
  - Language extensions
  - Language embeddings
- Theory
  - Advances in the rewriting calculus
  - Advances in rewriting logic

Program Committee

- Mark van den Brand, TU Eindhoven, Netherlands
- Horatiu Cirstea, IUT Nancy Charlemagne, France
- Steve Eker, SRI International, USA
- Maribel Fernandez, King's College London, UK
- Jeffrey G. Gray, University of Alabama at Birmingham, USA
- Berthold Hoffmann, University of Bremen, Germany
- Gunter Kniesel, University of Bonn, Germany
- Ralf Lammel, Microsoft, USA
- Pierre-Etienne Moreau, INRIA, France
- Dan Resler, Virginia Commonwealth University, USA
- Eelco Visser, Delft University of Technology, Netherlands
- Joost Visser (co-chair), Universidade do Minho in Braga, Portugal
- Victor Winter (co-chair), University of Nebraska at Omaha, USA

Proceedings

Accepted papers will be published in the preliminary proceedings volume, which will be available during the workshop. The final proceedings are expected to be published in Electronic Notes in Theoretical Computer Science (ENTCS), Elsevier.

Submissions

Papers (of at most 15 pages) should be submitted electronically via the web-based submission site. Any problems with the submission procedure should be reported to one of the PC chairs: Joost Visser (joost.visser@di.uminho.pt) or Victor Winter (vwinter@mail.unomaha.edu).

Important Dates

Sunday 25th March, 2007	Deadline for electronic submission of papers
Sunday 6th May, 2007	Notification of acceptance of papers
Sunday 20th May, 2007	Deadline for final versions of accepted papers
Friday 29th June, 2007	Workshop

Calculemus

Linz, Austria, June 27-30, 2007

http://www.risc.uni-linz.ac.at/about/conferences/Calculemus2007/

General

Calculemus is a series of conferences dedicated to the integration of computer algebra systems (CAS) and automated deduction systems (ADS) towards the development of universal mathematical assistant systems (MAS).

Currently, symbolic computation is divided into several (more or less) independent branches, traditional ones (e.g. computer algebra and theorem proving) as well as newly emerging ones (on user interfaces, knowledge management, theory exploration, etc.). The main concern of the Calculemus community is to bring these developments together in order to facilitate the theory, design, and implementation of integrated MAS that will routinely be used by mathematicians, computer scientists, and engineers in their every-day business.

For the upcoming Calculemus meeting, which will be held jointly with MKM2007 in Hagenberg, Austria, we seek original research papers in this context.

Scope

The scope of Calculemus covers all aspects of developing mathematical assistant systems, in particular, the interplay of automated reasoning and computer algebra. Potential areas of interest are:

- Automated reasoning in computer algebra
- Computer algebra in automated reasoning
- Interdisciplinary systems
- Infrastructure for mathematical services
- Theory exploration techniques
- Theory, design, and implementation of MAS
- Case studies and applications of MAS

Keynote Speakers

Thomas Hales, University of Pittsburgh
John Harrison, Intel Inc.
Peter Paule, RISC-Linz

Important Dates

February 12, 2007: Submission deadline
March 12, 2007: Notification of acceptance
March 26, 2007: Camera ready copies due
June 27--30, 2007: Calculemus 2007 in Hagenberg, Austria

Submission

Please submit your full paper of at most 12 pages prepared with the standard LNCS class style as pdf or ps file via **http://www.easychair.org/CALCULEMUS07/**
on or before February 12, 2007. Detailed formating instructions can be found on the Calculemus website.

Proceedings

Accepted papers will be published in the LNAI series of Springer.

Program Committee

- Alessandro Armando (DIST, Italy)
- Christoph Benzmiller (University of Cambridge, UK)
- Olga Caprotti (University of Helsinki, Finland)
- Jacques Carette (McMaster, Canada)
- Timothy Daly (Carnegie Mellon, USA)
- William M. Farmer (McMaster, Canada)
- Keith O. Geddes (Waterloo, Canada)
- Tom Hales (Pittsburgh, USA)
- Hoon Hong (North Carolina State University, USA)
- Deepak Kapur (New Mexico, USA)
- Manuel Kauers (RISC-Linz, Austria, Chair)
- Laura Kovacs (RISC-Linz, Austria)
- Petr Lisonek (Simon Fraser University, Canada)
- Roy McCasland (University of Edinburgh, UK)
- Renauld Rioboo (Universtite Pierre et Marie Curie, France)
- Volker Sorge (University of Birmingham, UK)
- Klaus Sutner (Carnegie Mellon, USA)
- Thomas Sturm (University of Passau, Germany)
- Wolfgang Windsteiger (RISC-Linz, Austria, Chair)

Workshops

The conference will be accompanied by several satellite workshops, which are currently under negotiation. Watch out for up-to-date information on our website.

Coordination Models and Languages

Paphos, Cyprus, June 6-8, 2007

http://www.discotec07.cs.ucy.ac.cy/

Modern information systems rely increasingly on combining concurrent, distributed, real-time, reconfigurable and heterogeneous components. New

models, architectures, languages, and verification techniques are necessary to cope with the complexity induced by the demands of today's software development. COORDINATION aims to explore the spectrum of languages, middleware, services, and algorithms that separate behavior from interaction, therefore increasing modularity, simplifying reasoning, and ultimately enhancing software development.

Topics of interest:

- PROGRAMMING LANGUAGE techniques that support orchestration and control of distributed and concurrent interaction.
- MIDDLEWARE ARCHITECTURES: shared spaces, publish-subscribe, event-based.
- DYNAMIC SOFTWARE ARCHITECTURES: software composition and scripting languages, dynamic software evolution and update, configuration and deployment languages.
- DEPENDABLE, RESOURCE-AWARE, REAL-TIME and EMBEDDED system coordination. Models and Foundations: component composition, verification, management of security and dynamic aspects of coordination.
- WEB SERVICES: Service-oriented Architectures, Workflow Systems. Programming abstractions for decentralized distributed systems such as P2P, mobile ad-hoc and sensor networks.
- TYPE SYSTEMS and SPECIFICATION LANGUAGES appropriate for coordination of concurrent systems.
- CASE STUDIES from E-Commerce, Factory Automation, Collaboration, Command and Control, or other systems.

PROGRAM COMMITTEE

Nadia Busi University of Bologna, IT
Vinny Cahill Trinity, IE
Paolo Ciancarini University of Bologna, IT
William Cook University of Texas, Austin, US
John Field IBM, US
Chris Gill Washington University, US
Aniruddha Gokhale Vanderbilt, US
Chris Hankin Imperial College, UK
Mike Hicks University of Maryland, US
Valerie Issarny INRIA, FR
Christoph Kirsch University of Salzburg, AT

Doug Lea SUNY Oswego, US
Toby Lehman IBM, US
Alberto Montresor University of Trento, IT
Amy L. Murphy ITC-IRST, IT & U. of Lugano, CH (Co-chair)
Oscar Nierstrasz University of Bern, CH
Anna Philippou University of Cyprus, CY
Ernesto Pimentel University of Malaga, ES
Giovanni Russello Imperial College, UK
Jan Vitek Purdue University, US (Co-chair)
Jim Waldo SUN Microsystems, US
Herbert Wiklicky Imperial College, UK

PROCEEDINGS

Proceedings of previous editions of this conference were published by Springer, in the Lecture Notes in Computer Science (LNCS) series and are available as LNCS volumes 1061, 1282, 1594, 1906, 2315, 2949, 3454 and 4038. Our intention is to continue this series.
Selected papers from COORDINATION will be invited to a special issue of The Science of Computer Programming journal.
A best student paper award will be given at the conference. To be eligible for consideration indicate on your submission if one or more of the paper's authors are students.

SUBMISSION INSTRUCTIONS

Authors are invited to submit full papers electronically in PDF before 27 January 2007. Further instructions are available from the conference web site.
Submissions must be formatted according to the LNCS guidelines (see http://www.springer.de/comp/lncs/authors.html) and must not exceed 17 pages in length (including all supplementary material). Papers that are not in the requested format or exceed the mandated length will be rejected without going through the review process. Simultaneous or similar submissions to other conferences or journals are not allowed.

IMPORTANT DATES

- Submission of papers: 27 January 2007
- Notification of acceptance: 7 March 2007
- Conference: 6-8 June 2007

Argumentation and Non-Monotonic Reasoning

Tempe, Arizona, May 14-16, 2007

http://lia.deis.unibo.it/confs/ArgNMR/

Aims and Scope

Research on Argumentation and Nonmonotonic Reasoning began in full force in the early eighties. The first attempts showed how argumentation results in a very natural way of conceptualizing Commonsense Reasoning, appropriately reflecting its defeasible nature. Further work in the KR&R community has shown that argumentation provides a useful perspective for relating different nonmonotonic formalisms. More recently, argumentation has been revealed as a powerful conceptual tool for exploring the theoretical foundations of reasoning and interaction in Autonomous Agents and Multiagent Systems.

This workshop will represent an opportunity for exchanging ideas on the fundamental theoretical basis and the design and implementation of argument-based systems including semantics, proof theory, applications to epistemic and practical reasoning, and the comparison of those systems with other types of nonmonotonic reasoning.

Topics

We solicit unpublished papers that present work on argumentation and nonmonotonic reasoning. We will privilege articles who emphasize connections between them. Relevant topics include, but are not limited to, the following:

- argumentation theories and logical foundations
- argumentation and logic programming
- formal models of argument
- semantics of argumentation
- operational semantics and execution models of argumentation systems
- argumentation and commonsense reasoning
- argumentation for practical reasoning and deliberation
- argumentation tools and applications
- argumentation for reasoning in multiagent systems

- argumentation dialogues in multiagent systems
- nonmonotonic reasoning in multiagent systems
- argumentation for legal reasoning
- argumentation and nonmonotonic reasoning in the semantic web
- implementations of argumentation systems

Important Dates

- Submission: 8 February 2007
- Notification: 12 March 2007
- Camera-ready: 29 March 2007
- ArgNMR: 14-16 May 2007 (one day)

Submissions

We welcome and encourage the submission of high quality, original papers, which have not been accepted for publication nor are currently under review for another journal or conference. Papers should be written in English, formatted according to the Springer LNCS style (http://www.springer.com/comp/lncs/Authors.html), and they should not exceed sixteen (16) pages including title page, figures, references, etc.

Proceedings and post-workshop publications

A printed volume with the proceedings will be available at the workshop. The proceedings of ArgNMR are also planned to form the basis for publishing a post-workshop volume, and/or a special issue of an international journal, subject to appropriate quality.

Programme Committee

- Leila Amgoud, IRIT-CNRS Toulouse, France
- Grigoris Antoniou, FORTH-ICS, Greece
- Pietro Baroni, U Brescia, Italy
- Trevor J. Bench-Capon, U Liverpool, United Kingdom
- Carlos Iván Chesñevar, U Nacional del Sur, Bahia Blanca, Argentina
- Jürgen Dix, TU Clausthal, Germany
- Phan Minh Dung, Asian Institute of Technology, Thailand
- Lluis Godo, IIIA-CSIC, Spain
- Anthony Hunter, U College London, United Kingdom

- Antonis C. Kakas, U Cyprus
- Gabriele Kern-Isberner, U Dortmund, Germany
- Nicolas Maudet, U Paris-Dauphine, France
- Peter McBurney, U Liverpool, United Kingdom
- Donald Nute, U. Georgia, Athens, GE, United States
- Henry Prakken, U Utrecht, U Groningen, The Netherlands
- Iyad Rahwan, British U Dubai, UAE & U Edinburgh, United Kingdom
- Tran Cao Son, New Mexico State U, NM, United States
- Francesca Toni, Imperial College London, United Kingdom

Organization

Guillermo R. Simari, U. Nacional del Sur, Bahia Blanca, Argentina
Paolo Torroni, U. Bologna, Italy

Correspondence and Equivalence of Nonmonotonic Theories

Tempe, Arizona, May 14-16, 2007

http://www.kr.tuwien.ac.at/cent2007/

The systematic study of intertheory relations such as strong and uniform equivalence has recently become an active sub-area of research in the field of LPNMR. Various kinds of correspondence relations that may hold between logic programs or between nonmonotonic theories have been analysed and shown to be of practical relevance for theory or program transformation, optimisation and modularity. Several systems for verifying such relations have been implemented. Different types of knowledge representation and reasoning tasks have begun to be explored in this context, such as abductive and inductive reasoning, causal reasoning, preference-based reasoning or reasoning about updates.

In the field of KRR more generally one notes an increased interest in intertheory relations that are relevant for ontologies, eg to describe modular ontologies or equivalences between ontologies or their parts. It may therefore be of interest to combine work in this area with work on equivalences between nonmonotonic rules. We are also interested in new results on equivalences between different ontology languages proposed for the Semantic Web, particularly in combinations with (nonmonotonic) rules. Frameworks for study might therefore include e.g. DL-programs or hybrid knowledge bases that provide combinations of a classical or

description logic KB with logic programming rules.

The scope of the workshop covers all aspects of the study and application of intertheory relations in the LPNMR area. In particular it welcomes contributions that

- extend the catalogue of useful relations or provide novel characterisations
- characterise relations wrt different approaches to LP and NMR
- examine specialised reasoning tasks, eg planning, diagnosis, explanation, reasoning about actions, reasoning about ontologies
- explore practical applications
- present system descriptions and comparisons

Workshop topics

Workshop topics include, but are not limited to:

- logical characterisations
- applications
- computational complexity
- implementation issues
- benchmarks and system comparisons
- relations to datalog and database theory
- relations to ontologies and Semantic Web languages

Submission and Presentation Format

Papers must be written in English and we encourage both original research papers or system descriptions. Submissions must not exceed twelve (12) pages including title page, references and figures, and must be formatted according to the Springer LNCS/LNAI authors' instructions, but also shorter papers will be considered. For system presentations a length of 4 pages is recommended. We will use easychair for your electronic submissions, the submission page is accessible at:

http://www.easychair.org/CENT2007/

Important Dates

- 23 Feb 2007, Submission of papers
- 30 March 2007, Notification of acceptance

- 20 April 2007, Camera-ready versions due
- 14-16 May 2007, Workshop

Committees

Steering committee:
David Pearce
Axel Polleres
Agustin Valverde
Stefan Woltran
Programme Committee:
Wolfgang Faber
Katsumi Inoue
Vladimir Lifschitz
Fangzhen Lin
Emilia Oikarinen
Riccardo Rosati
Hans Tompits

Conference on Theory and Applications of Satisfiability Testing

Lisbon, Portugal, May 28-31, 2007

http://sat07.ecs.soton.ac.uk

The International Conference on Theory and Applications of Satisfiability Testing is the primary annual meeting for researchers studying the propositional satisfiability problem (SAT). SAT'07 is the tenth SAT conference. SAT'07 features the SAT competition, the QBF competition, the Pseudo-Boolean evaluation, and the MAX-SAT evaluation.

SCOPE

Many hard combinatorial problems can be encoded into SAT. Therefore improvements on heuristics on the practical, as well as theoretical insights into SAT apply to a large range of real-world problems. More specifically, many important practical verification problems can be rephrased as SAT problems. This applies to verification problems in hardware and software. Thus SAT is becoming one of the most important core technologies to verify secure and

dependable systems. The topics of the conference span practical and theoretical research on SAT and its applications and include but are not limited to proof systems, proof complexity, search algorithms, heuristics, analysis of algorithms, hard instances, randomized formulae, problem encodings, industrial applications, solvers, simplifiers, tools, case studies and empirical results. SAT is interpreted in a rather broad sense: besides propositional satisfiability, it includes the domain of quantified boolean formulae (QBF), constraints programming techniques (CSP) for word-level problems and their propositional encoding and particularly satisfiability modulo theories (SMT).

SUBMISSION

Submissions should contain original material and can either be regular research papers up to 14 pages or short papers up to 6 pages. Double submissions including submissions as short and long papers will be rejected. Submissions should use the Springer LNCS style. All appendices, tables, figures and the bibliography must fit into the page limit. Submissions deviating from these requirements may be rejected without review. All accepted papers including short papers will be published in the proceedings of the conference. The conference proceedings will be published within Springer LNCS series. The submission page is http://www.easychair.org/SAT2007. Papers have to be submitted electronically as PDF files. Paper submissions are due by January 19.

PROGRAM CHAIRS

Joao Marques-Silva, University of Southampton, UK
Karem Sakallah, University of Michigan, USA

LOCAL CHAIR

Ines Lynce, Technical University of Lisbon, Portugal

INVITED SPEAKERS

Martin Davis, New York University, USA
Andrei Voronkov, University of Manchester, UK

IMPORTANT DATES

January 19, Paper Submission

March 2, Author Notification
March 16, Final Version

TECHNICAL PROGRAM COMMITTEE

Fahiem Bacchus, University of Toronto, Canada
Paul Beame, University of Washington, USA
Armin Biere, Johannes Kepler University, Austria
Adnan Darwiche, UCLA, USA
Leonardo de Moura, Microsoft Research, USA
Niklas Een, Cadence Design Systems, USA
John Franco, University of Cincinnati, USA
Ziyad Hanna, Intel Corp., USA
Ian Gent, University of St. Andrews, UK
Enrico Giunchiglia, Universita di Genova, Italy
Carla Gomes, Cornell University, USA
Aarti Gupta, NEC Research Labs, USA
Edward A. Hirsch, Steklov Inst. of Mathematics, Russia
Joonyoung Kim, Intel Corp., USA
Hans Kleine-Buning, Univ. Paderborn, Germany
James Kukula, Synopsys ATG, USA
Oliver Kullmann, University of Wales Swansea, UK
Daniel Le Berre, Universite d'Artois, France
Chu-Min Li, Universite de Picardie, France
Ines Lynce, Technical University of Lisbon, Portugal
Panagiotis Manolios, Georgia Institute of Technology, USA
Vasco Manquinho, Technical University of Lisbon, Portugal
Slawomir Pilarski, Magma DA, USA
Steve Prestwich, University College Cork, Ireland
Roberto Sebastiani, Universita di Trento, Italy
Hossein Sheini, CMU, USA
Laurent Simon, Universite Paris Sud, France
Ewald Speckenmeyer, Universitat Koln, Germany
Ofer Strichman, Technion, Israel
Stefan Szeider, Durham University, UK
Armando Tacchella, Universita di Genova, Italy
Allen Van Gelder, UC Santa Cruz, USA
Hans van Maaren, Technische Universiteit Delft, Netherlands
Toby Walsh, National ICT, Australia
Lintao Zhang, Microsoft Research, USA

SAT COMPETITION

http://www.satcompetition.org/2007

Daniel Le Berre, Universite d'Artois, France
Laurent Simon, Universite Paris Sud, France
Ewald Speckenmeyer, Universitat Koln, Germany
Geoff Sutcliffe, University of Miami, USA
Lintao Zhang, Microsoft Research, USA

QBF COMPETITION

http://www.qbflib.org/qbfeval

Massimo Narizzano, Universita di Genova, Italy
Luca Pulina, Universita di Genova, Italy
Armando Tacchella, Universita di Genova, Italy

PSEUDO BOOLEAN EVALUATION

http://www.cril.univ-artois.fr/PB07

Olivier Roussel, Universite d'Artois, France
Vasco Manquinho, Technical University of Lisbon, Portugal

MAX-SAT EVALUATION

http://www.maxsat07.udl.es

Josep Argelich, IIIA-CSIC, Spain
Chu Min Li, Universite de Picardie, France
Felip Manya, IIIA-CSIC, Spain
Jordi Planes, IIIA-CSIC, Spain

Workshop on Programming Multi-Agent Systems

Hawaii, USA, May 14-18, 2007

http://www.cs.uu.nl/ProMAS/

Even though the contributions of the multi-agent systems (MAS) community can make a significant impact in the development of open distributed systems, the techniques resulting from such contributions will only be widely adopted when suitable programming languages and development tools are available. Furthermore, such languages and tools must incorporate those techniques in a principled but practical way, so as to support the ever more complex task of professional programmers, in particular when the systems have to operate in dynamic environments.

The ProMAS workshop series aims to address the theoretical and practical programming issues related to developing and deploying multi-agent systems. In particular, ProMAS aims to address how multi-agent systems designs or specifications can be effectively implemented. In its previous editions, ProMAS constituted an invaluable occasion bringing together leading researchers from both academia and industry to discuss issues on the design of programming languages and tools for multi-agent systems. In particular, the workshop promotes the discussion and exchange of ideas concerning the techniques, concepts, requirements, and principles that are important for multi-agent programming technology.

We encourage the submission of proposals for programming languages and development tools that provide specific programming constructs to facilitate the implementation of the essential concepts used in multi-agent system analysis and specifications (e.g., mental attitudes, distribution, and social interaction). We also welcome submissions describing significant multi-agent applications, as well as agent programming tools that allow the integration of agents with legacy systems. Further, we are particularly interested in approaches or applications that show clearly the added-value of multi-agent programming, and explain why and how this technology should be adopted by designers and programmers both in academia and industry.

Specific topics for this workshop include, but are not limited to:

- Programming Languages for multi-agent systems
- Extensions of traditional languages for multi-agent programming
- Theoretical and practical aspects of multi-agent programming
- Computational complexity of MAS
- Semantics for multi-agent programming languages
- High-level executable multi-agent specification languages
- Algorithms, techniques, or protocols for multi-agent issues (e.g.,

coordination, cooperation, negotiation)
- Agent communication issues in multi-agent programming
- Implementation of social and organisational aspects of MAS
- Formal methods for specification and verification of MAS
- Verification tools for implementations of MAS
- Test and debugging tools and techniques
- Agent development tools and platforms
- Generic tools and infrastructures for multi-agent programming
- Interoperability and standards for MAS
- Programming mobile agents
- Safety and security for mobile MAS deployment
- Fault tolerance and load balancing for mobile MAS
- Application areas for multi-agent programming languages
- Applications using legacy systems
- Programming MAS for Grid-based applications
- Programming MAS for the Semantic Web
- Deployed (industrial-strength) MAS
- Benchmarks and testbeds for comparing MAS languages and tools

Important Dates:

Paper submission deadline: 5 February, 2007
Notifications of acceptance/rejection: 5 March, 2007
Camera-ready copies due: 19 March, 2007
Workshop Date: 14th/15th May, 2007 (TBA)

Submission Details:

Authors can submit their papers via a conference management system, available at the following address:

http://confs.deis-ce.unibo.it/ProMAS07

First, you will be asked to register into the system, then you will get a user id and a password that you can use to submit your paper. Papers should be formatted using Springer LNCS style (**http://www.springer.de/comp/lncs/authors.html**) and have a maximum of 15 pages.
Accepted papers will be published as a technical report and distributed among participants during the workshop. As was the case for previous editions of the ProMAS workshop, we are planning to publish extended versions

of selected papers as a volume of the Lecture Notes in Computer Science series by Springer-Verlag.

Organising Committee:

Mehdi Dastani (Utrecht University, The Netherlands) **http://www.cs.uu.nl/~mehdi**
Amal El Fallah Seghrouchni (University of Paris VI, France) **http://www-poleia.lip6.fr/~elfallah/**
Alessandro Ricci (DEIS, Universita' di Bologna, Italy) **http://lia.deis.unibo.it/~ari**
Michael Winikoff (RMIT University, Australia) **http://www.cs.rmit.edu.au/~winikoff**

Steering Committee:

Rafael Bordini (University of Durham, UK)
Juergen Dix (Clausthal University of Technology, Germany)

European Semantic Web Conference

Innsbruck, Austria, June 3-7, 2007

http://www.eswc2007.org/

The vision of the Semantic Web is to enhance today's web via the exploitation of machine-processable meta data. The explicit representation of the semantics of data, enriched with domain theories (Ontologies), will enable a web that provides a qualitatively new level of service. It will weave together a large network of human knowledge and makes this knowledge machine-processable. Various automated services will help the users to achieve their goals by accessing and processing information in machine-understandable form. This network of knowledge systems will ultimately lead to truly intelligent systems, which will be employed for various specialized reasoning subsystems to accomplish complex tasks. Many technologies and methodologies are being developed within Artificial Intelligence, Natural Language Processing, Machine Learning, Databases, Multimedia Systems, Distributed Systems, Software Engineering and Information Systems that can contribute towards the realization of this vision.

The 4th Annual European Semantic Web Conference (ESWC 2007) will present

the latest results in research and application of Semantic Web technologies, including knowledge mark-up languages, Semantic Web services, and ontology management. ESWC 2007 will also feature a special industry-oriented event, a forum for gaining a better understanding of these new technologies and their business aspects. The conference will offer a tutorial program to get up to speed with European and global developments in this exciting new area.

Several distinguished scientists will give an invited talk at the conference; among them, prof. Stefano Ceri (Tech. Univ. of Milan, Italy), prof. Georg Gottlob (Oxford Univ., UK), prof. Ning Zhong (Maebashi Institute of Technology, Japan).

ESWC 2007 is sponsored by ESSI - a group of European Commission 6th Framework Programme projects. Together these projects aim to improve world-wide research and standardisation in the area of the Semantic Web. For more information on ESSI, please visit www.essi-cluster.org.

Submissions

ESWC 2007 welcomes the submission of excellent original research and application papers dealing with all aspects of the Semantic Web, particularly those related to the subject areas indicated by the topics below. We particularly encourage the submission of papers on industrial efforts and experiences with Semantic Web projects. We encourage theoretical, methodological, empirical, and applications papers.

The proceedings of this conference will be published in Springer's Lecture Notes in Computer Science series. Paper submission and reviewing for ESWC 2007 will be electronic, via the conference Web site: **http://www.eswc2007.org/**. Papers, due 15 December, 2006, should not exceed fifteen (15) pages in Springer LNCS format.

Papers may be accepted as (i) full papers, or as (ii) short papers with poster presentation.

Important Dates

Abstract Submission: 8 December, 2006
Full Paper Submission: 15 December, 2006
Notification: 26 February, 2007
Camera-Ready Papers due: 16 March, 2007
Conference: 3 - 7 June, 2007

Conference Topics of Interest

Topics of interest to the conference include (but are not restricted to):

- Ontology Management (creation, evolution, evaluation, etc.)
- Ontology Alignment (mapping, matching, merging, mediation and reconciliation)
- Ontology Learning and Metadata Generation (e.g. HLT and ML approaches)
- Multimedia and Semantic Web
- Semantic Annotation of Data
- Semantic Web Trust, Privacy, Security and Intellectual Property Rights
- Semantic Web Rules and Query Languages
- Logics for the Semantic Web
- Reasoning on the Semantic Web
- Behavior in the Semantic Web
- Searching, Querying, Visualizing, Navigating and Browsing the Semantic Web
- Personalization and User Modelling
- User Interfaces and Semantic Web
- Semantic Grid and Middleware
- Semantic Web Services (description, discovery, invocation, composition, choreography, etc.)
- Semantic Web-based Knowledge Management (e.g. Semantic Desktop, Knowledge Portals)
- Semantic Web for e-Business, e-Culture, e-Government, e-Health, e-Learning, e-Science
- Database Technologies for the Semantic Web
- Data Semantics and Web Semantics
- Semantic Interoperability
- Semantic Workflows
- Semantic Web Mining

We particularly welcome application papers which clearly show benefits of Semantic Web technologies in practical settings.

General Chair

Enrico Franconi (Free University of Bozen-Bolzano, Italy), franconi@inf.unibz.it

Program Chairs

Michael Kifer (State University of New York at Stony Brook, USA), kifer@cs.stonybrook.edu
Wolfgang May (Georg-August-Universität Göttingen, Germany), may@informatik.uni-goettingen.de

International Colloquium on Automata, Languages and Programming

Wroclaw, Poland, July 9-13, 2007

http://icalp07.ii.uni.wroc.pl/

The 34th International Colloquium on Automata, Languages and Programming, the main conference and annual meeting of the European Association for Theoretical Computer Science EATCS will take place from the 9th to the 13th of July 2007 in Wroclaw, Poland. This year the conference will be colocated with 22nd Annual IEEE Symposium on Logic in Computer Science (LICS 2007) and Logic Colloquium 2007.

Following the successful experience of the last two years, ICALP 2007 will complement the established structure of the scientific program based on Track A on Algorithms, Automata, Complexity and Games, and Track B on Logic, Semantics, and Theory of Programming, corresponding to the two main streams of the journal Theoretical Computer Science, with a special Track C on Security and Cryptography Foundations. The aim of Track C is to allow a deeper coverage of a particular topic, selected for each year's edition of ICALP on the basis of its timeliness and relevance for the theoretical computer science community.

Papers presenting original research on all aspects of theoretical computer science are sought. Typical but not exclusive topics of interest are:

- Track A - Algorithms, Automata, Complexity and Games (PC Chair: Lars Arge, University of Aarhus, Denmark)
  - Algorithmic Aspects of Networks
  - Algorithmic Game Theory
  - Automata Theory

- Combinatorics in Computer Science
- Computational Biology
- Computational Complexity
- Computational Geometry
- Data Structures
- Design and Analysis of Algorithms
- Internet Algorithmics
- Machine Learning
- Parallel, Distributed and External Memory Computing
- Quantum Computing

- Track B - Logic, Semantics, and Theory of Programming (PC Chair: Andrzej Tarlecki, University of Warsaw, Poland)
  - Algebraic and Categorical Models
  - Automata and Formal Languages
  - Emerging and Non-standard Models of Computation
  - Databases, Semi-Structured Data and Finite Model Theory
  - Principles of Programming Languages
  - Logics, Formal Methods and Model Checking
  - Models of Concurrent, Distributed, and Mobile Systems
  - Models of Reactive, Hybrid and Stochastic Systems
  - Program Analysis and Transformation
  - Specification, Refinement and Verification
  - Type Systems and Theory, Typed Calculi

- Track C - Security and Cryptography Foundations (PC Chair: Christian Cachin, IBM Zurich Research Laboratory, Switzerland)
  - Cryptographic Notions, Mechanisms, Systems and Protocols
  - Cryptographic Proof Techniques, Lower bounds, Impossibilities
  - Foundations of Secure Systems and Architectures
  - Logic and Semantics of Security Protocols
  - Number Theory and Algebraic Algorithms in Cryptography
  - Pseudorandomness, Randomness, and Complexity Issues
  - Secure Data Structures, Storage, Databases and Content
  - Security Modeling: Combinatorics, Graphs, Games, Economics
  - Specifications, Verifications and Secure Programming
  - Theory of Privacy and Anonymity
  - Theory of Security in Networks and Distributed Computing
  - Quantum Cryptography and Information Theory

SUBMISSION GUIDELINES

Authors are invited to submit an extended abstract presenting original research. The abstract should not exceed 10 single-spaced pages including title and abstract, but excluding bibliography and appendices, should be in single-column format, use at least 11-point fonts, and have reasonable margins. If the authors believe that more details are essential to substantiate the main claims of the paper, they may include a clearly marked appendix that will be read at the discretion of the program committee. Submissions deviating significantly from these guidelines risk rejection without consideration of their merits.

Submissions should indicate to which track (A, B, or C) the paper is submitted. No simultaneous submission to other publication outlets (either a conference or a journal) is allowed.

PROCEEDINGS

The proceedings will be published in the Lecture Notes in Computer Science Series by Springer-Verlag. The final version of each accepted paper must be submitted in electronic form conforming to the LNCS style and not exceeding 12 pages.

IMPORTANT DATES

- Submission: January 25, 2007
- Notification: April 5, 2007
- Final version due: April 23, 2007
- Deadline for workshop proposals: November 30, 2006
- Notification for workshop proposals: December 17, 2006

Computer Aided Verification
Berlin, Germany, July 3-7, 2007

http://www.cav2007.org/

Aims and Scope:

CAV'07 is the 19th in a series dedicated to the advancement of the theory and practice of computer-aided formal analysis methods for hardware and software systems. CAV considers it vital to continue its leadership in hardware verification, and maintain its recent momentum in software verification. The conference covers the spectrum from theoretical results to concrete applications, with an emphasis on practical verification tools and the algorithms and techniques that are needed for their implementation. The proceedings of the conference will be published in the Springer-Verlag Lecture Notes in Computer Science series. A selection of papers will be invited to a special issue of the International Journal on Formal Methods and System Design.
Topics of interest include:

- Algorithms and tools for verifying models and implementations
- Hardware verification techniques
- Hybrid systems and embedded systems verification
- Program analysis and software verification
- Modeling and specification formalisms
- Deductive, compositional, and abstraction techniques for
- rification
- Testing and runtime analysis based on verification technology
- Applications and case studies
- Verification in industrial practice

Paper submission:

There are two categories of submissions:

1. Regular papers. Submissions, not exceeding thirteen (13) pages using Springer's LNCS format, should contain original research, and sufficient detail to assess the merits and relevance of the contribution. For papers reporting experimental results, authors are strongly encouraged to make their data available with their submission. Submissions reporting on case studies in an industrial context are strongly invited, and should describe details, weaknesses and strength in sufficient depth. Simultaneous submission to other conferences with proceedings or submission of material that has already been published elsewhere is not allowed.
2. Tool presentations. Submissions, not exceeding four (4) pages using Springer's LNCS format, should describe the implemented tool and its novel

features. A demonstration is expected to accompany a tool presentation. Papers describing tools that have already been presented in this conference before will be accepted only if significant and clear enhancements to the tool are reported and implemented.

Information concerning the procedure for submissions will be available on the conference home page:

http://www.cav2007.org

Submissions will be evaluated by the program committee for inclusion in the proceedings, which will be published by Springer-Verlag in the LNCS series. Papers exceeding the stated maximum length or submitted after January 28, 2007 run the risk of rejection without review.
On an experimental basis for this year, authors will be granted access to the text content of their reviews during the review process. Authors will be given a short time period in which to submit feedback, which may (at the PC's discretion) be taken into account in the decision process. Strict guidelines on length and content of feedback will be provided to the authors.

Important dates:

Paper submission (firm): January 28, 2007
Author feedback period: March 9-11, 2007
Notification of acceptance: March 23, 2007
Final version due: April 20, 2007

Program Chairs:

Werner Damm, U Oldenburg, damm at informatik.uni-oldenburg.de
Holger Hermanns, Saarland U, hermanns at cs.uni-sb.de

Program Committee:

Parosh Abdullah, Uppsala U
Rajeev Alur, U Penn
Sergey Berezin, Synopsis
Armin Biere, JKU Linz
Roderick Bloem, TU Graz
Ahmed Bouajjani, U Paris 7
Alessandro Cimatti, IRST Trento
Edmund M. Clarke, CMU

Werner Damm, CvO U Oldenburg
E Allen Emerson, U Texas (tbc.)
Limor Fix, Intel
Patrice Godefroid, Microsoft Research
Ganesh Gopalakrishnan, U of Utah
Susanne Graf, Verimag
Orna Grumberg, Technion
Holger Hermanns, Saarland U
Robert Jones, Intel
Orna Kupferman, Hebrew U
Robert Kurshan, Cadence
John Lygeros, ETH Zuerich (tbc.)
Tom Melham, Oxford U
Ken McMillan, Cadence
Jakob Rehof, U Dortmund
Koushik Sen, UC Berkeley
Fabio Somenzi, U Boulder
Ashish Tiwari, SRI International
Frits Vaandrager, U Nijmegen
Yaron Wolfstal, IBM Haifa

Workshop on Software Engineering for Answer Set Programming

Tempe, Arizona, May 14-16, 2007

Over the last ten years Answer Set Programming (ASP) has grown from a pure theoretical knowledge representation and reasoning formalism to a computational approach with a very strong formal backing. At present, ASP is seen as the computational embodiment of non-monotonic reasoning incorporating techniques of databases, knowledge representation, logic and constraint programming. ASP has become an appealing tool for knowledge representation and reasoning and thanks
to the increasing efficiency of the implementations of ASP solvers, the field has now started to tackle the first industrially relevant applications.

Writing complex programs in any language is not an easy task, with ASP being no exception. Most of the modern popular programming languages have an abundance of tools and development methodologies to facilitate and improve the

coding process. Given the differences in for example language design, execution, and application domains for languages such as Java and C++, the existing methodologies and tools that are available are mostly not suitable for ASP. Therefore development tools and software engineering methodologies specifically designed for ASP are required.

This workshop aims to bring together researchers who are currently working on or are interested in the development of dedicated tools, techniques, and methodologies to facilitate the development of answer set programs.

Topics

Authors are invited to submit original research or system description papers on software engineering tools or techniques for answer set programming.
The list of topics of interest includes but is not limited to:

- Modeling tools
- (Domain Specific) Front and/or Back-ends
- Methodologies
- Debuggers
- (Graphical) User Interfaces
- Integrated Development Environment (IDE)
- Software engineering metrics

Workshop co-chairs

Marina De Vos, University of Bath, UK (**mdv@cs.bath.ac.uk**)
Torsten Schaub, University of Potsdam, (**torsten@cs.uni-potsdam.de**)

Program Committee

Tommi Syrjanen, Helsinki University of Technology, Finland
Enrico Pontelli, New Mexico State University, US
Tran Cao Son, New Mexico State University, US
Stefan Woltran, Technical University of Vienna, Austria
Martin Brain, University of Bath, UK
Richard Watson, Texas Tech University, US
Wolfgang Faber, University of Calabria, Italy
Ken Satoh, National Institute of Informatics, Japan
Yan Zhang, University of Western Sydney, Australia

Submission Details

Submitted articles will undergo peer-review. The paper must be in Springer LNCS format and must not exceed 15 pages in total. Submission should be sent as a pdf to both workshop chairs. Formal paper proceedings will be available during the conference and will also be published online.

Important Dates

Paper Submission: 8 February 2006
Paper Acceptance/Rejection Notification: 12 March 2006
Camera Ready Papers: 20 April 2006

Modeling and Using Context

Roskilde University, Denmark, August 20-24, 2007

http://context-07.ruc.dk/

The Sixth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT'07) will provide a forum for presenting and discussing high-quality research and applications on context. The conference will include paper and poster presentations, system demonstrations, workshops, and a doctoral consortium. The conference invites researchers and practitioners to share insights and cutting-edge results from a wide range of disciplines including:

- Computer Science, especially Artificial Intelligence and Ubiquitous Computing
- Cognitive Science
- Linguistics
- Organizational Sciences
- Philosophy
- Psychology
- Application areas such as Medicine and Law

Context affects a wide range of activities in humans and animals as well as in artificial agents and other systems. The importance of context is widely acknowledged, and "context" has become an area of study in its own right, as

evidenced by numerous workshops, symposia, seminars, and conferences on this area. CONTEXT, the oldest conference series focusing on context, is unique in its emphasis on interdisciplinary research. Previous CONTEXT conferences have been held in Rio de Janeiro, Brazil (CONTEXT'97), Trento, Italy (CONTEXT'99, LNCS 1688), Dundee, Scotland (CONTEXT'01, LNCS 2116), Palo Alto, U.S.A. (CONTEXT'03, LNCS 2680), and Paris, France (CONTEXT'05, LNCS 3554). Each of these brought together researchers and practitioners from many disparate fields to discuss and report on context-related research and projects.

TOPICS OF INTEREST

The following list illustrates sample research areas whose perspectives on context are solicited for the conference. This is not an exhaustive list, and contributions addressing context from other perspectives are welcome. The conference scope includes the contextual issues related to areas such as:

- Analogy and Case-Based Reasoning
- Intelligent/Semantic Web Systems
- Autonomous Agents and Agent-based Systems
- Knowledge Engineering and Management
- Knowledge Representation
- Cognitive Modeling
- Language Understanding and Production
- Concepts and Categorization
- Learning
- Context-Aware Applications and Systems
- Memory, Representation and Access
- Multiagent Systems and Interagent Communication
- Databases
- Distributed Information Systems
- Neuroscience
- Formal Semantics and Pragmatics
- Formal Theories of Context
- Ontology Management
- Heterogeneous Information Integration
- Organizational Theory and Design
- Perception
- Human Decision-Making and Decision
- Philosophical Foundations of Context
- Support Systems

- Problem Solving and Planning
- Human-Centered Computing
- Reasoning
- Human-Computer Interaction
- Relevance Computation and Relevance Theories
- Information Management
- Intelligent Tutoring Systems
- Situated and Distributed Cognition
- Intelligent User Interfaces
- Ubiquitous Computing

CONFERENCE EVENTS

CONTEXT'07 will include paper presentation sessions, a poster and demonstration session, two days of workshops, and a doctoral consortium. Workshops and the doctoral consortium will circulate separate calls for papers and participation, which will also be available at the conference web site.

ACCEPTANCE CRITERIA AND SUBMISSION CATEGORIES

Because CONTEXT'07 will be an interdisciplinary forum, all submissions will be evaluated both for their technical merit and for their accessibility to an interdisciplinary audience. Works that transcend disciplinary boundaries are especially encouraged.

Submissions may be for full papers, poster abstracts, or demonstration abstracts. Full papers will be accepted either for oral presentation or for presentation at a poster session. All accepted full paper submissions will be published in the proceedings. Accepted posters and demonstrations will be presented at the poster session, and the associated abstracts will be published in a brochure distributed to attendees. For additional details see the conference web site.

For a paper to appear in the proceedings, at least one author must register for the conference by the deadline for camera-ready copy.

SUBMISSION PROCEDURES

Papers must be submitted electronically as PDF files. Submissions cannot exceed 14 pages in the Springer LNAI format. Detailed formatting and submissions instructions, as well as LaTeX and Word templates, will be available in the author instructions section of the conference Web site.

All accepted authors will have the option of presenting a system demonstration at the poster session. Authors wishing to present a demonstration without an accompanying paper must submit a demonstration abstract. Demonstration abstracts should describe cutting-edge systems not described in paper submissions. Demonstration abstracts should summarize the system's behavior and significance, and should include at least one screen shot. If desired, they may also include the URL of an informal video on the web. Demonstration abstracts should be at most 2 pages long.

MULTIPLE SUBMISSIONS POLICY

CONTEXT'07 will not accept any paper which, at the time of submission, is under review for or has already been published or accepted for publication in a journal or another conference. This restriction does not apply to submissions for workshops and other venues with a limited audience.

IMPORTANT DATES

Deadline for workshop proposal submissions January 31, 2007
Deadline for paper submissions March 15, 2007
Deadline for poster and demonstration abstract submissions March 15, 2007
Notification of acceptance/rejection for paper submissions May 7, 2007
Suggested deadline for workshop paper submission May 15, 2007
Deadline for final versions of accepted papers May 31, 2007
Workshop days August 20-21, 2007
Main conference (including poster and demo sessions) August 22-24, 2007

CONFERENCE ORGANIZERS

CONFERENCE CHAIR
Boicho Kokinov, New Bulgarian University, Bulgaria

PROGRAM CO-CHAIRS
Daniel C. Richardson, UCSC, USA
Thomas R. Roth-Berghofer, DFKI, Germany
Laure Vieu, IRIT-CNRS, France, and ISTC-CNR, Italy

WORKSHOPS CHAIR
Stefan Schulz, The e-Spirit Company GmbH, Dortmund, Germany

ORGANIZING COMMITTEE

Henning Christiansen, Roskilde University, Denmark (Chair)
Troels Andreasen, Roskilde University, Denmark
John Gallagher, Roskilde University, Denmark
Mads Rosendahl, Roskilde University, Denmark
Jørgen Villadsen, Technical University of Denmark (Publicity Chair)

STEERING COMMITTEE

Chiara Ghidini, ITC-irst, Italy (Chair)
Varol Akman, Bilkent University, Turkey
Massimo Benerecetti, University of Naples, Italy
Paolo Bouquet, University of Trento, Italy
Patrick Brézillon, University of Paris 6, France
Anind Dey, Carnegie Mellon University, USA
Fausto Giunchiglia, ITC-irst, Italy
Boicho Kokinov, New Bulgarian University, Bulgaria
David Leake, Indiana University, USA
Luciano Serafini, Trentino Cultural Institute (ITC), Italy
Rich Thomason, University of Michigan, USA
Roy Turner, University of Maine, USA
Roger A. Young, University of Dundee, UK

FOR ADDITIONAL INFORMATION

Please see **http://context-07.ruc.dk**/ for additional information on the conference, complete committee information, and contacts for questions.

Henning Christiansen
professor, ph.d.
Computer Science Section, Bldg. 42.1
Roskilde University
P.O.Box 260, DK-4000 Roskilde, DENMARK

Coordinating Agents' Plans and Schedules

Honolulu, Hawaii, May 14-15, 2007

http://www.st.ewi.tudelft.nl/~mathijs/caps07

Description

Multiagent planning is concerned with planning by (and for) multiple agents. Nowadays a major issue in multiagent planning is the coordination of single-agent planners. Here, coordination is studied not only during the execution of plans, but also in the (pre)-planning phase.

A wide range of real applications could benefit from such coordinated planning technology, for example, in transportation and logistics, health care management, space missions, military tasks, and disaster management. Also, planning in the context of human-computer (or human-robot) interaction is inherently a multiagent planning task. Coordinating the plans of the involved entities up front has the potential to improve the efficiency of the whole system. However, currently, a great amount of research seems to focus solely on either planning, or the coordination of agents without the context of a plan.

The purpose of this workshop is to address the problems that arise when coordinating the plans and schedules of multiple agents. We therefore solicit papers with original work, as well as position statements or surveys that relate to one or more of the following questions:

- Which applications require decentralized planning?
    - Can we derive benchmark problems from these applications?
- How can we evaluate multiagent planning techniques?
    - How to measure communication costs, privacy loss, flexibility and robustness?
    - How to measure plan quality when agents arevself-interested (e.g., multi-objectivevoptimization, or game theoretical concepts such asvPareto optimal solutions)?
- What are efficient techniques to deal with the many problems inherent to a dynamic and uncertain multiagent world?
    - How to deal with local autonomy, privacy issues, and conflicting preferences?
    - How to deal with uncertainty and incomplete information?
    - How to coordinate multiagent plan diagnosis and (local) plan repair?
    - How to coordinate plans when agents' objectives (tasks, intentions, preferences,...) evolve overtime?

Paper should clarify their relevance to these questions.
To summarize, specific topics of interest include (butvare not limited to):

- multiagent planning and scheduling applications
- strategies for testing/evaluating distributedvplan/schedule management techniques
- self-interested planning agents
- privacy in distributed planning
- game theoretic planning
- managing local autonomy in team planning/scheduling
- mixed initiative and adjustable autonomy in distributed planning/scheduling
- negotiation over tasks/intentions in distributed planning/scheduling
- distributed continual planning/scheduling
- plan/schedule maintenance in single and multiagent systems
- plan/schedule repair in stochastic and adversarial domains
- active (distributed) monitoring to trigger plan/schedule maintenance
- distributed planning under uncertainty
- multiagent planning with sparse or unreliable communication

Paper submissions

Authors are encouraged to submit papers or position statements electronically in PDF format. Submitted papers should be formatted according to ACM specifications. ACM style guides, as well as templates and style sheets for Microsoft Word, WordPerfect, and LaTeX can be found at the ACM webpage. (**http://www.acm.org/sigs/pubs/proceed/template.html**)
Papers should be no more than 8 pages. Please submit your paper at the workshop website no later than February 5, 2007.

Accepted papers will be distributed as informal working notes, printed copies of which will be available at the workshop. Depending on the quality of the submissions, we are planning to select a subset of the papers, and give the authors of these papers the opportunity to publish a revised version of their workshop paper in post-proceedings (e.g. by IOS Press or Springer).

Important dates

- Deadline for submissions: February 5, 2007
- Notifications: March 5, 2007
- Deadline for camera-ready copy: March 19, 2007
- Workshop: half a day at May 14th or 15th, 2007

Program committee

Organizers

- Michael Brenner, Albert-Ludwigs-Universität Freiburg (brenner at informatik.uni-freiburg.de)
- Brad Clement, Jet Propulsion Laboratory, Pasadena (bclement at jpl.nasa.gov)
- Mathijs de Weerdt, Delft University of Technology (M.M.deWeerdt at tudelft.nl)

Program committee

- Anthony Barrett, Jet Propulsion Laboratory, Pasadena
- Keith Decker, University of Delaware
- Ed Durfee, University of Michigan
- Boi Faltings, Swiss Federal Institute of Technology
- Piotr Gmytrasiewicz, University of Illinois at Chicago
- Nick Hawes, University of Birmingham
- Sven Koenig, University of Southern California
- Roman van der Krogt, University College Cork
- Victor Lesser, University of Massachusetts
- Karen Myers, SRI International
- Jeff Rosenschein, Hebrew University of Jerusalem
- Reid Simmons, Carnegie Mellon University
- Steve Smith, Carnegie Mellon University
- Tom Wagner, DARPA
- Cees Witteveen, Delft University of Technology
- Shlomo Zilberstein, University of Massachusetts

International Conference on Automated Deducation

Bremen, Germany, July 17-20, 2007

http://www.cadeconference.org/meetings/cade21

CADE is the major forum for the presentation of research in all aspects of automated deduction.

- Logics of interest include propositional, first-order, equational, higher-order, classical, intuitionistic, constructive, modal, temporal, many-valued, substructural, description, and meta-logics, logical frameworks, type theory and set theory.
- Methods of interest include resolution, tableaux, term rewriting, induction, unification, constraint solving, SAT solving, decision procedures, saturation, model generation, model checking, natural deduction, sequent calculi, proof planning, proof presentation, proof checking, and explanation.
- Applications of interest include hardware and software development, systems analysis and verification, deductive databases, functional and logic programming, computer mathematics, natural language processing, computational linguistics, robotics, planning, knowledge representation, and other areas of AI.

Paper submission:

Submission is electronic in PostScript or PDF format via the EasyChair system. Submitted papers must conform to the Springer LNCS style, preferrably using LaTeX2e and the Springer llncs class files. Submissions can be full papers, for work on foundations, applications, or implementation techniques (15 pages), as well as system descriptions (5 pages), for describing publicly available systems. The proceedings will be published in the Springer LNCS series. For further information and submission instructions, see http://www.cadeconference.org/meetings/cade21

Important dates:

Submission of title and abstract: February 16, 2007
Submission papers: February 23, 2007
Notification of acceptance: April 16, 2007
Final version due: May 11, 2007
Workshops and tutorials: July 15-16, 2007
Conference: July 17-20, 2007

Conference Chair: Michael Kohlhase (IUB)
Workshop and Tutorial Chair: Christoph Benzmueller (Saarland Univ)
Program Chair: Frank Pfenning (CMU)

Program Committee:

David Basin ETH Zuerich

Christoph Benzmueller Cambridge University
Maria Paola Bonacina Universita degli Studi di Verona
Simon Colton Imperial College London
Gilles Dowek Ecole Polytechnique
Rajeev Gore Australian National University
Jean Goubault-Larrecq ENS Cachan
Reiner Haehnle Chalmers University of Technology
John Harrison Intel Corporation
Michael Kohlhase International University Bremen
Dale Miller INRIA-Futurs and Ecole Polytechnique
Tobias Nipkow Technical University Munich
Hans de Nivelle MPII Saarbruecken
Albert Oliveras Technical University of Catalonia
Frank Pfenning (chair) Carnegie Mellon University
Ulrike Sattler University of Manchester
Manfred Schmidt-Schauss University of Frankfurt
Cesare Tinelli University of Iowa
Andrei Voronkov University of Manchester
Toby Walsh National ICT Australia and Univ of New South Wales

Papers to appear in TPLP and TOCL

Contents

Theory and Practice of Logic Programming

http://www.cs.kuleuven.ac.be/~dtai/projects/ALP/TPLP/index.html

Volume 6, Issue 4, July 2006

- Epistemic Foundation of Stable Model Semantics, Yann Loyer and Umberto Straccia. pp 355-393
- Computing minimal models, stable models and answer sets, Zbigniew Lonc and Miroslaw Truszczynski. pp 395-449
- EPspectra: A Formal Toolkit for Developing DSP Software Applications, Hahnsang Kim, Thierry Turletti, Amar Bouali. pp 451-481

Volume 6, Issue 5, September 2006

- Programming Finite-Domain Constraint Propagators in Action Rules, Neng-Fa Zhou. pp 483-508
- A three-valued semantics for logic programmers, Lee Naish. pp 509-538
- Temporal Phylogenetic Networks and Logic Programming, Esra Erdem, Vladimir Lifschitz, and Don Ringe. pp 539-558
- Planning with Preferences using Logic Programming, Tran Cao Son and Enrico Pontelli. pp 559-608

Special Issues (to appear)

Special Issue on Multiparadigm Languages and Constraint Programming

- Introduction to the special issue on multiparadigm languages and constraint programming, Moreno Falaschi and Michael Maher,
- Mapping fusion and synchronized hyperedge replacement into logic programming, Ivan Lanese and Ugo Montanari
- A comparison between two logical formalisms for rewriting, Miguel Palomino
- Combining relational algebra, SQL, constraint modelling, and local search, Marco Cadoli and Toni Mancini,
- Constraint-based automatic verification of abstract models of multitreaded programs, Giorgio Delzanno
- Removing redundant arguments automatically, Maria Alpuente, Santiago Escobar, and Salvador Lucas,
- Forward slicing of functional logic programs by partial evaluation, Josep Silva and German Vidal,
- Demand Analysis with Partial Predicates, Julio Marino, Angel Herranz and Juan Jose Moreno-Navarro
- Integration of Declarative and Constraint Programming, Petra Hofstedt and Peter Pepper

Accepted Regular Papers

- Set Unification, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi.
- Improving PARMA Trailing, Tom Schrijvers, Maria Garcia de la Banda, Bart Demoen, Peter J. Stuckey
- Incremental copying garbage collection for WAM-based Prolog systems, Ruben Vandeginste, Bart Demoen.
- Embedding Defeasible Logic into Logic Programming, Grigoris Antoniou, David Billington, Guido Governatori and Michael J. Maher.
- Intelligent search strategies based on adaptive Constraint Handling Rules, Armin Wolf.
- A Knowledge-Based Approach for Selecting Information Sources, Thomas Eiter, Michael Fink, and Hans Tompits
- Constraint Functional Logic Programming over Finite Domains, Antonio J. Fernandez, Teresa Hortala-Gonzalez, Fernando Saenz-Perez and Rafael del Vado-Virseda.
- Reasoning and Planning with Sensing Actions, Incomplete Information, and Static Causal Laws using Answer Set Programming, Phan Huy Tu, Tran Cao Son, and Chitta Baral

- Well-founded and Stable Semantics of Logic Programs with Aggregates, Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe
- PALS: Efficient or-Parallelism on Beowulf clusters, Enrico Pontelli, Karen Villaverde, Hai-Feng Guo, Gopal Gupta
- Decomposable Theories, Khalil Djelloul
- Logic programs with monotone abstract constraint atoms, Victor Marek, lkka Niemela, and Mirek Truszczynski
- Automated verification of weak equivalence within the smodels system, Tomi Janhunen and Emilia Oikarinen
- Calculating modules in contextual logic program refinement, Robert Colvin, Ian J. Hayes and Paul Strooper
- Updates in Answer Set Programming: An Approach Based on Basic Structural Properties, Mauricio Osorio and Victor Cuevas

Accepted Technical Note

- Logic programming with default, weak and strict negations, Susumu Yamasaki.
- Fast Frequent Querying with Lazy Control Flow Compilation, Remko Troncon, Gerda Janssens, Bart Demoen, Henk Vandecasteele
- A Constructive Semantic Characterization of Aggregates in Answer Set Programmino, Tran Cao Son, Enrico Pontelli

Accepted Programming Pearl

Accepted Book Review

- Explanatory Nonmonotonic Reasoning by Alexander Bochman World Scientific, Hardback: ISBN 981-256-101-3 Victor W. Marek

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 7, Number 4 (October 2006)

- Domain-Dependent Knowledge in Answer Set Planning Tran Cao Son, Chitta Baral, Nam Tran, and Sheila McIlraith
- Defining Functions on Equivalence Classes Larry Paulson
- Extensional Equivalence and Singleton Types Christopher A. Stone and Robert Harper
- Efficient Solving of Quantified Inequality Constraints over the Real Numbers Stefan Ratschan
- The Strength of Replacement in Weak Arithmetic Stephen Cook and Neil Thapen
- Splitting an Operator: Algebraic Modularity Results for Logics with Fixpoint Semantics Joost Vennekens, David Gilis and Marc Denecker
- Kleene Algebra with Domain J. Desharnais, B. Möller, G. Struth

Volume 8, Number 1 (January 2007) (tentative)

- A System of Interaction and Structure Alessio Guglielmi
- Compilability of Propositional Abduction Paolo Liberatore and Marco Schaerf
- Results on the Quantitative Mu-Calculus Annabelle McIver and Carroll Morgan
- On Compositionality and its Limitations Alex Rabinovich
- Logical Characterizations of Heap Abstractions G. Yorsh, T. Reps, M. Sagiv and R. Wilhelm
- Abstract Canonical Inference Maria Paola Bonacina and Nachum Dershowitz

Volume 8, Number 2 (April 2007) (tentative)

- Sound and Complete Elimination of Singleton Kinds Karl Crary
- Recycling Computed Answers in Rewrite Systems for Abduction Fangzhen Lin and Jia-Huai You
- Where Fail-Safe Default Logics Fail Paolo Liberatore

- Logical Definability and Query Languages over Ranked and Unranked Trees M. Benedikt, L. Libkin and F. Neven
- On Unification for Bounded Distributive Lattices Viorica Sofronie-Stokkermans
- The Arithmetical Complexity of Dimension and Randomness John M. Hitchcock, Jack H. Lutz, Sebastiaan A. Terwijn

Future Issues (the order of the papers can change)

- PELCR: Parallel Environment for Optimal Lambda Calculus Reduction M. Pedicini, F. Quaglia
- Ordinary Interactive Small-Step Algorithms II Andreas Blass and Yuri Gurevich
- Ordinary Interactive Small-Step Algorithms III Andreas Blass and Yuri Gurevich
- Semantical Characterizations and Complexity of Equivalences in Answer Set Programming Thomas Eiter, Michael Fink, and Stefan Woltran (Electronic Appendix)
- Paraconsistent Reasoning and Preferential Entailments by Signed Quantified Boolean Formulae Ofer Arieli
- The Axiomatic Translation Principle for Modal Logic Renate A. Schmidt and Ullrich Hustadt
- Probabilistic Abstraction for Model Checking: An Approach Based on Property Testing Sophie Laplante, Richard Lassaigne, Frederic Magniez, Sylvain Peyronnet and Michel de Rougemont
- First-Order Queries on Structures of Bounded Degree Are Computable with Constant Delay Arnaud Durand and Etienne Grandjean
- A Sequent Calculus and a Theorem Prover for Standard Conditional Logics Nicola Olivetti, Gian Luca Pozzato and Camilla Schwind
- Removing Propagation Redundant Constraints in Redundant Modeling Chiu Wo Choi, Jimmy Ho-Man Lee and Peter J. Stuckey
- Probabilistic Interval XML Edward Hung, Lise Getoor and V.S. Subrahmanian
- A Game-Based Framework for CTL Counterexamples and 3-Valued Abstraction-Refinement Sharon Shoham and Orna Grumberg (Electronic Appendix)
- A Formally Verified Proof of the Prime Number Theorem Jeremy Avigad,

Kevin Donnelly, David Gray and Paul Raff

- Polymorphic Type Inference for the Named Nested Relational Calculus Jan Van den Bussche and Stijn Vansummeren
- Predicate Abstraction with Indexed Predicates Shuvendu Lahiri and Randal Bryant
- Verifying Nondeterministic Probabilistic Channel Systems Against Omega-Regular Linear-Time Properties Christel Baier, Nathalie Bertrand and Philippe Schnoebelen
- A Concrete Framework for Environment Machines Malgorzata Biernacka and Olivier Danvy
- Outlier Detection by Logic Programming Fabrizio Angiulli, Gianluigi Greco and Luigi Palopoli (Electronic Appendix)
- A Comprehensive Combination Framework Silvio Ghilardi, Enrica Nicolini and Daniele Zucchelli
- Coordination in Answer Set Programming Chiaki Sakama and Katsumi Inoue
- Alternating Timed Automata Slawomir Lasota and Igor Walukiewicz
- Bounds on the Automata Size for Presburger Arithmetic Felix Klaedtke
- Durations and Parametric Model-Checking in Timed Automata Véronique Bruyère, Emmanuel Dall'Oli and Jean-Francois Raskin
- First-order Complete and Computationally Complete Query Languages for Spatio-Temporal Databases Sofie Haesevoets, Floris Geerts and Bart Kuijpers
- A Logic for Non-Monotone Inductive Definitions Marc Denecker and Eugenia Ternovska
- 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

Accepted Papers in Logic Programming-related Conferences

Contents

International Symposium on Practical Aspects of Declarative Languages

Nantes, France, January 14-15, 2007

http://www.informatik.uni-kiel.de/~mh/padl07

Accepted Papers

- Ricardo Rocha. *On Improving the Efficiency and Robustness of Table Storage Mechanisms for Tabled Evaluation*
- Chuck Liang. *Aspect-Oriented Programming in Higher-Order and Linear Logic*
- Takeshi Morimoto, Yasunao Takano and Hideya Iwasaki. *Instantly Turning a Naive Exhaustive Search into Three Efficient Searches with Pruning*
- Michael Eichberg, Matthias Kahl, Diptikalyan Saha, Mira Mezini and Klaus Ostermann. *Automatic Incrementalization of Prolog based Static Analyses*
- Claudio Russo. *The Joins Concurrency Library*
- Elvira Albert, Miguel Gómez-Zamalloa, Laurent Hubert and Germán Puebla. *Verification of Java Bytecode using Analysis and Transformation of Logic Programs*
- Pablo Berdaguer, Alcino Cunha, Hugo Pacheco and Joost Visser. *Coupled Schema Transformation and Data Conversion for XML and SQL*
- Karl Klose, Klaus Ostermann and Michael Leuschel. *Partial Evaluation of*

Pointcuts

- Jens Fisseler, Gabriele Kern-Isberner, Christoph Beierle, Andreas Koch and Christian Müller. *Algebraic Knowledge Discovery using Haskell*
- Liwen Huang, Paul Hudak and John Peterson. *HPorter: Using Arrows to Compose Parallel Processes*
- Beata Sarna-Starosta and C.R. Ramakrishnan. *Compiling Constraint Handling Rules for Efficient Tabled Evaluation*
- Edison Mera, Pedro Lopez-Garcia, German Puebla, Manuel Carro and Manuel Hermenegildo. *Combining Static Analysis and Profiling for Estimating Execution* Times
- Per Gustafsson and Konstantinos Sagonas. *Applications, Implementation and Performance Evaluation of Bit Stream Programming in Erlang*
- Alan Bond. *BAD, a declarative language for brain modeling*
- Chongbinbg Liu and Enrico Pontelli. *Inductive Logic Programming by Instance Patterns*
- Duncan Coutts, Don Stewart and Roman Leshchinskiy. *Rewriting Haskell Strings*
- Vitor Santos Costa. *Prolog Performance on Larger Datasets*
- Reza Rafeh, Maria Garcia de la Banda, Kimbal Marriott and Mark Wallace. *From Zinc to Design Model*
- Andreas Podelski and Andrey Rybalchenko. *ARMC: a logical choice for software model checking with abstraction refinement*

Symposium on Theoretical Aspects of Computer Science
Aachen, Germany, February 22-24, 2007

http://www-i7.informatik.rwth-aachen.de/stacs07/

Accepted Papers

- Manindra Agrawal, Thanh Minh Hoang and Thomas Thierauf. *The polynomially bounded perfect matching problem is in NC^2*
- Alberto Bertoni, Massimiliano Goldwurm and Violetta Lonati. *On the complexity of unary tiling-recognizable picture languages*.
- Dietmar Berwanger. *Admissibility in infinite games*
- Laurent Bienvenu. *Kolmogorov-Loveland stochasticity and Kolmogorov complexity*

- Francine Blanchet-Sadri, Joshua Gafni and Kevin Wilson. *Correlations of Partial Words*
- Hans Bodlaender. *A Cubic Kernel for Feedback Vertex Set*
- Mikolaj Bojanczyk and Piotr Hoffman. *Reachability in Unions of Commutative Rewriting Systems is Decidable*
- Felix Brandt, Felix Fischer and Markus Holzer. *Symmetries and the Complexity of Pure Nash Equilibrium*
- Janina Brenner and Guido Schaefer. *Cost Sharing Methods for Makespan and Completion Time Scheduling*
- Davide Bresolin, Angelo Montanari and Pietro Sala. *An optimal tableau-based decision algorithm for Propositional Neighborhood Logic*
- Peter Buergisser. *On defining integers in the counting hierarchy and proving arithmetic circuit lower bounds*
- Sergiu Bursuc, Hubert Comon and Stephanie Delaune. *Associative-Commutative Deducibility Constraints*
- Costas Busch and Srikanta Tirthapura. *A Deterministic Algorithm for Summarizing Asynchronous Streams over a Sliding Window*
- Jin-Yi Cai and Pinyan Lu. *On Symmetric Signatures in Holographic Algorithms*
- Ioannis Caragiannis. *Wavelength management in WDM rings to maximize the number of connections*
- Olivier Carton, Jean Berstel, Luc Boasson and Isabelle Fagnot. *A First Investigation of Sturmian trees*
- Arkadev Chattopadhyay, Andreas Krebs, Michal Koucky, Mario Szegedy, Pascal Tesson and Denis Therien. *Languages with bounded multiparty communication complexity*
- Andrew Childs, Aram Harrow and Pawel Wocjan. *Weak Fourier-Schur sampling, the hidden subgroup problem, and the quantum collision problem*
- Amin Coja-Oghlan, Michael Krivelevich and Danny Vilenchik. *Almost all k-colorable graphs are easy*
- Bruno Courcelle and Andrew Twigg. *Compact Forbidden-set Routing*
- Artur Czumaj and Christian Sohler. *Small Space Representations for Metric Min-Sum k-Clustering and their Applications*
- Peter Damaschke. *The Union of Minimal Hitting Sets: Parameterized Combinatorial Bounds and Counting*
- Benjamin Doerr. *Randomly Rounding Rationals with Cardinality Constraints and Derandomizations*
- Adrian Dumitrescu and Csaba Toth. *Light Orthogonal Networks with Constant Geometric Dilation*
- Robert Elsässer and Thomas Sauerwald. *Broadcasting vs. Mixing and*

Information Dissemination on Cayley Graphs

- Javier Esparza, Stefan Kiefer and Michael Luttenberger. *On Fixed Point Equations over Commutative Semirings*
- Eldar Fischer and Orly Yahalom. *Testing Convexity Properties of Tree Colorings*
- Enrico Formenti and Petr Kurka. *A search algorithm for the maximal attractor of a cellular automaton*
- Iftah Gamzu and Danny Segev. *Improved Online Algorithms for the Sorting Buffer Problem*
- Hugo Gimbert. *Pure stationary optimal strategies in Markov decision processes*
- Christian Glaßer, Alan L. Selman, Stephen Travers and Klaus W. Wagner. *The Complexity of Unions of Disjoint Sets*
- Iman Hajirasouliha, Hossein Jowhari, Ravi Kumar and Ravi Sundaram. *On Completing Latin Squares*
- Masahito Hayashi, Kazuo Iwama, Harumichi Nishimura, Rudy Raymond and Shigeru Yamashita. *Quantum Network Coding*
- Pinar Heggernes, Karol Suchan, Ioan Todinca and Yngve Villanger. *Characterizing minimal interval completions: Towards better understanding of profile and pathwidth*
- Chien-Chung Huang. *Cheating to Get Better Roommates in a Random Stable Matching*
- Christian Hundt and Maciej Liskiewicz. *On the Complexity of Affine Image Matching*
- Gábor Ivanyos, Miklos Santha and Luc Sanselme. *An efficient quantum algorithm for the hidden subgroup problem in extraspecial groups*.
- Telikepalli Kavitha, Kurt Mehlhorn and Dimitrios Michail. *New Approximation Algorithms for Minimum Cycle Bases of Graphs*
- Pascal Koiran and Sylvain Perifel. *VPSPACE and a transfer theorem over the reals*
- Daniel Kral. *Computing representations of matroids of bounded branch-width*
- Ralf Kuesters and Tomasz Truderung. *On the Automatic Analysis of Recursive Security Protocols with XOR*
- Manfred Kunde. *A new bound for pure greedy hot potato routing*
- Gregory Lafitte and Michael Weiss. *Universal Tilings*
- Soeren Laue and Stefan Funke. *Bounded-hop energy-efficient Broadcast in low-dimensional Metrics via Coresets*
- Troy Lee. *A new rank technique for formula size lower bounds*
- Nutan Limaye, Meena Mahajan and Raghavendra Rao. *Arithmetizing*

Classes arround NC1 and L
- Sylvain Lombardy. *On the Size of the Universal Automaton of a Regular Language*
- Ilan Newman and Yuri Rabinovich. *Hard Metrics from Cayley Graphs of Abelian Groups*
- Jan Poland. *On the Consistency of Discrete Bayesian Learning*
- Andrea Sattler-Klein. *An Exponential Lower Bound For Prefix Gröbner Bases in Free Monoid Rings*
- Henning Schnoor and Ilka Schnoor. *Enumerating all Solutions for Constraint Satisfaction Problems*
- Lutz Schröder and Dirk Pattinson. *Rank-1 Modal Logics are Coalgebraic*
- Thomas Schwentick and Volker Weber. *Bounded-Variable Fragments of Hybrid Logics*
- Hans Ulrich Simon. *A Characterization of Strong Learnability in the Statistical Query Model*
- Marc Tedder and Derek Corneil. *An Optimal, Edges-Only Fully Dynamic Algorithm for Distance-Hereditary Graphs*
- Oleg Verbitsky. *Planar graphs: Logical complexity and parallel isomorphism tests*

Workshop on Non-monotonic Reasoning, Action, and Change
Hyderabad, India, January 7-8, 2007

http://research.it.uts.edu.au/magic/NRAC/2007/

Accepted Papers

- *Why the monkey needs the box: a serious look at a toy domain,* Selim T Erdo•an, Paolo Ferraris, Vladimir Lifschitz, Wanwan Ren
- *A general purpose framework for approximate and cost-effective simulation in commonsense reasoning systems,* Benjamin Johnston, Mary-Anne Williams
- *Reification of action instances in the Leonardo calculus,* Erik Sandewall
- *Reasoning about actions with description logics,* Conrad Drescher, Michael Thielscher
- *Integrating reasoning about actions and Bayesian networks,* Yves Martin, Michael Thielscher
- *FIPA communicative acts in defeasible logic,* Guido Boella, Guido

Governatori, Joris Hulstijn
- *On the existence of answer sets in normal extended logic programs,* Martin Caminada, Chiaki Sakama
- *Repeated negotiation of logic programs,* Wu Chen, Mingyi Zhang, Norman Foo
- *The cause and treatments of floating conclusions and zombie paths,* Yi Mao, Beihai Zhou
- *Towards autonomous strategy decisions in the RoboCup four-legged league,* Michael Quinlan, Oliver Obst, Stephan Chalup
- *Automatic construction of a heuristic search function for general game playing,* Stephan Schiffel, Michael Thielscher
- *Plans in cooperation logics: a modular approach,* Jelle Gerbrandy, Luigi Sauro
- *On domain-independent heuristics for planning with qualitative preferences,* Jorge Baier, Sheila McIlraith
- *A novel framework for plan recognition: planning graph as a basis,* Minghao Yin, Jigui Sun, Dunbo Cai, Shuai Lu
- *Modular basic action theories,* Yilan Gu, Mikhail Soutchanski
- *Iterated belief change via prime implicates,* Maurice Pagnucco
- *On iterated revision of total preorders—preliminary results,* Richard Booth, Thomas Meyer
- *Extending conceptual graphs for representing partial knowledge,* Madalina Croitoru, Ernesto Compatangelo

International Joint Conference on Artificial Intelligence
Hyderabad, India, January 9-12, 2007

http://www.ijcai-07.org/?q=index.html

Accepted Papers

225	On the Logic of Normative Systems	Thomas Ågotnes, Wiebe van der Hoek, Juan A. Rodríguez-Aguilar, Carles Sierra, Michael Wooldridge
233	Quantified Coalition Logic	Thomas Ågotnes, Wiebe van der Hoek, Michael Wooldridge

223	The Mathematical Morpho-Logical View on Reasoning about Space	Marco Aiello, Brammert Ottens
972	AWA* . A Window Constrained Anytime Heuristic Search Algorithm	Sandip Aine, Partha P. Chakrabarti, Rajeev Kumar
546	Detecting Stochastically Scheduled Activities in Video	Massimiliano Albanese, Vincenzo Moscato, Antonio Picariello, V.S. Subrahmanian, Octavian Udrea
448	An Axiomatic Approach to Personalized Ranking Systems	Alon Altman, Moshe Tennenholtz
618	Keep the Decision Tree and Estimate the Class Probabilities using its Decision Boundary	Isabelle Alvarez, Stephan Bernard, Guillaume Deffuant
480	Solving POMDPs Using Quadratically Constrained Linear Programs	Christopher Amato, Daniel S. Bernstein, Shlomo Zilberstein
1017	Market Based Resource Allocation with Incomplete Information	Bo An, Chunyan Miao, Zhiqi Shen
887	Updates for Nonlinear Discriminants	Edin Andelic, Martin Schafföner, Marcel Katz, Sven E. Krüger, Andreas Wendemuth
495	The Logic Behind Weighted CSP	Carlos Ansótegui, María Luisa Bonet, Jordi Levy, Felip Manyà
564	Distributed Data Mining: Why Do More Than Aggregating Models	Mohamed Aoun-Allah, Guy Mineau
604	An Information-Theoretic Analysis of Memory Bounds in a Distributed Resource Allocation Mechanism	Ricardo M. Araujo, Luis C. Lamb
650	A Description Logic of Change	Alessandro Artale, Carsten Lutz, David Toman
1775	Effective Control Knowledge Transfer Through Learning Skill and Representation Hierarchies	Mehran Asadi, Manfred Huber

594	From Generic Knowledge to Specific Reasoning for Medical Image Interpretation using Graph based Representations	Jamal Atif, Céline Hudelot, Geoffroy Fouquier, Isabelle Bloch, Elsa Angelini
227	GUNSAT: A Greedy Local Search Algorithm for Unsatisfiability	Gilles Audemard, Laurent Simon
300	Symmetry Breaking in Quantified Boolean Formulae	Gilles Audemard, Saïd Jabbour, Lakhdar Saïs
296	Image Modeling using Tree Structured Conditional Random Fields	Pranjal Awasthi, Aakanksha Gagrani, Balaraman Ravindran
432	Completing Description Logic Knowledge Bases using Formal Concept Analysis	Franz Baader, Bernhard Ganter, Baris Sertkaya, Ulrike Sattler
1159	Stable Biclustering of Gene Expression Data with Nonnegative Matrix Factorizations	Liviu Badea, Doina Tilivea
298	A Fully Connectionist Model Generator for Covered First-Order Logic Programs	Sebastian Bader, Pascal Hitzler, Steffen Hölldobler, Andreas Witzel
1506	A Heuristic Search Approach to Planning with Temporally Extended Preferences	Jorge A. Baier, Fahiem Bacchus, Sheila A. McIlraith
1398	Determining Expert Profiles (With an Application to Expert Finding)	Krisztian Balog, Maarten de Rijke
571	Learning .Forgiving. Hash Functions: Algorithms & Large Scale Tests	Shumeet Baluja, Michele Covell
1462	General Game Learning using Knowledge Transfer	Bikramjit Banerjee, Peter Stone
647	Open Information Extraction from the Web	Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, Oren Etzioni
1600	Non-monotonic Temporal Logics for Goal Specification	Chitta Baral, Jicheng Zhao

1589	Using the Probabilistic Logic Programming Language P-log for Causal and Counterfactual Reasoning and Non-Naive Conditioning	Chitta Baral, Matt Hunsaker
1354	Computational Aspects of Analyzing Social Network Dynamics	Chris Barrett, Harry B. Hunt III, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, Richard E. Stearns, Mayur Thakur
1291	A Machine Learning Approach for Statistical Software Testing	Nicolas Baskiotis, Michèle Sebag, Marie-Claude Gaudel, Sandrine-Dominique Gouraud
276	Learning by Analogy: A Classification Rule for Binary and Nominal Data	Sabri Bayoudh, Laurent Miclet, Arnaud Delhay
295	Trust Based Recommender System for Semantic Web	Punam Bedi, Harmeet Kaur, Sudeep Marwaha
442	Visually Tracking Football Games Based on TV Broadcasts	Michael Beetz, Suat Gedikli, Jan Bandouch, Bernhard Kirchlechner, Nico von Hoyningen-Huene, Alexander Perzylo
1531	Interactive Clustering of Text Collections According to a User-Specified Criterion	Ron Bekkerman, Hema Raghavan, James Allan, Koji Eguchi
1429	Web Page Clustering using Heuristic Search in the Web Graph	Ron Bekkerman, Shlomo Zilberstein, James Allan
1679	Context-Driven Predictions	Marc G. Bellemare, Doina Precup
1452	QCSP Made Practical by Virtue of Restricted Quantification	Marco Benedetti, Arnaud Lallouet, Jérémie Vautard
1529	On the Compilation of Stratified Belief Bases under Linear and Possibilistic Logic Policies	Salem Benferhat, Safa Yahi, Habiba Drias
139	Waiting and Relocation Strategies in Online Stochastic Vehicle Routing	Russell Bent, Pascal Van Hentenryck
1355	Learning Implied Global Constraints	Christian Bessiere, Remi Coletta, Thierry Petit

1338	Query-Driven Constraint Acquisition	Christian Bessiere, Remi Coletta, Barry O.Sullivan, Mathias Paulin
725	Phonetic Models for Generating Spelling Variants	Rahul Bhagat, Eduard Hovy
348	Heuristic Selection of Actions in Multiagent Reinforcement Learning	Reinaldo A. C. Bianchi, Carlos H. C. Ribeiro, Anna H. R. Costa
1160	Unsupervised Discretization Using Kernel Density Estimation	Marenglen Biba, Floriana Esposito, Stefano Ferilli, Nicola Di Mauro, Teresa Maria Altomare Basile
1105	A General Framework for Scheduling in a Stochastic Environment	Julien Bidot, Thierry Vidal, Philippe Laborie, John Christopher Beck
259	Entailment Semantics for Rules with Priorities	David Billington
1191	Sequence Prediction Exploiting Similary Information	István Bíró, Zoltán Szamonek, Csaba Szepesvári
112	Coalitions in Action Logic	Stefano Borgo
84	Higher-Order Potentialities and their Reducers: A Philosophical Foundation Unifying Dynamic Modeling Methods	Tibor Bosse, Jan Treur
755	Fast Planning with Iterative Macros	Adi Botea, Martin Müller, Jonathan Schaeffer
809	New Constraint Programming Approaches for the Computation of Leximin-Optimal Solutions in Constraint Networks	Sylvain Bouveret, Michel Lema? tre
585	A Game-Theoretic Analysis of Strictly Competitive Multiagent Scenarios	Felix Brandt, Felix Fischer, Paul Harrenstein, Yoav Shoham
881	Spiteful Bidding in Sealed-Bid Auctions	Felix Brandt, Tuomas Sandholm, Yoav Shoham
723	Identifying Expressions of Opinion in Context	Eric Breck, Yejin Choi, Claire Cardie

1168	Mediating between Qualitative & Quantitative Representations for Task-Orientated Human-Robot Interaction	Michael Brenner, Nick Hawes, John Kelleher, Jeremy Wyatt
438	Contextual Default Reasoning	Gerhard Brewka, Floris Roelofsen, Luciano Serafini
1158	Case-based Multilabel Ranking	Klaus Brinker, Eyke Hüllermeier
762	Efficient and Robust Independence-Based Markov Network Structure Discovery	Facundo Bromberg, Dimitris Margaritis
238	Fast Image Alignment Using Anytime Algorithms	Rupert Brooks, Tal Arbel, Doina Precup
180	Planning for Gene Regulatory Network Intervention	Daniel Bryce, Seungchan Kim
---	Cooperating Reasoning Processes: More than Just the Sum of Their Parts	Alan Bundy
1445	Exploiting Known Taxonomies in Learning Overlapping Concepts	Lijuan Cai, Thomas Hofmann
141	Locality Sensitive Discriminant Analysis	Deng Cai, Xiaofei He, Kun Zhou, Han , Bao
536	EQL-Lite: Effective First-Order Query Processing in Description Logics	Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati
1349	Multi-Dimensional Bid Improvement Algorithm for Simultaneous Auctions	Teddy Candale, Sandip Sen
726	Detect and Track Latent Factors with Online Nonnegative Matrix Factorization	Bin Cao, Dou Shen, Jian-Tao Sun, Xuanhui Wang, Qiang Yang, Zheng Chen
1474	Learning Semantic Descriptions of Web Information Sources	Mark James Carman, Craig A. Knoblock
617	Using Linear Programming for Bayesian Exploration in Markov Decision Processes	Pablo Samuel Castro, Doina Precup

621	Bidding Languages and Winner Determination for Mixed Multi-unit Combinatorial Auctions	Jesús Cerquides, Ulle Endriss, Andrea Giovannucci, Juan A. Rodríguez-Aguilar
1307	Supervised Latent Semantic Indexing using Adaptive Sprinkling	Sutanu Chakraborti, Rahman Mukras, Robert Lothian, Nirmalie Wiratunga, Stuart Watt, David Harper
345	Coalitional Bargaining with Agent Type Uncertainty	Georgios Chalkiadakis, Craig Boutilier
681	Learning to Walk through Imitation	Rawichote Chalodhorn, David B. Grimes, Keith Grochow, Rajesh P.N. Rao
1683	An Improved Probabilistic Ant based Clustering for Distributed Databases	Ramachandran Chandrasekar, Thanukrishnan Srinivasan
776	Compiling Bayesian Networks Using Variable Elimination	Mark Chavira, Adnan Darwiche
237	Directed Graph Embedding	Mo Chen, Qiong Yang, Xiaoou Tang
1409	Long-Distance Mutual Exclusion for Propositional Planning	Yixin Chen, Zhao Xing, Weixiong Zhang
1471	Iterated Weaker-than-Weak Dominance	Shih-Fen Cheng, Michael P. Wellman
411	A Lattice-based Approach to Computing Warranted Beliefs in Skeptical Argumentation Frameworks	Carlos Iván Chesñevar, Guillermo Ricardo Simari
544	Reaching Envy-Free States in Distributed Negotiation Settings	Yann Chevaleyre, Ulle Endriss, Sylvia Estivie, Nicolas Maudet
497	Privacy and Artificial Agents, or, Is Google Reading My Email?	Samir Chopra, Laurence White
1247	Representing Kriegspiel States with Metapositions	Paolo Ciancarini, Gian Piero Favini
1237	Towards an Integration of Golog and Planning	Jens Claßen, Patrick Eyerich, Gerhard Lakemeyer, Bernhard Nebel

1428	Constructing Career Histories: A Case Study in Disentangling the Threads	Paul R. Cohen
799	Learning and Transferring Action Schemas	Paul R. Cohen, Yu-Han Chang, Clayton T. Morrison, Carole Beal
353	Incremental Mechanism Design	Vincent Conitzer, Tuomas Sandholm
926	A Framework for Decentralized Qualitative Model-Based Diagnosis	Luca Console, Claudia Picardi, Daniele Theseider Duprè
1169	Optimal Soft Arc Consistency	Martin C. Cooper, Simon de Givry, Thomas Schiex
154	Exploiting Independence in a Decentralised and Incremental Approach of Diagnosis	Marie-Odile Cordier, Alban Grastien
592	A Conceptual Graph approach for the generation of referring expressions	Madalina Croitoru, Kees Van Deemter
1261	Online Learning and Exploiting Relational Models in Reinforcement Learning	Tom Croonenborghs, Jan Ramon, Hendrik Blockeel, Maurice Bruynooghe
936	A Logical Framework for Modularity of Ontologies	Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, Ulrike Sattler
1540	When is Temporal Planning Really Temporal?	William Cushing, Subbarao Kambhampati, Mausam, Daniel S. Weld
1523	Permanents, Transport Polytopes and Positive Definite Kernels on Histograms	Marco Cuturi
1470	Utile Distinctions for Relational Reinforcement Learning	William Dabney, Amy McGovern
1040	Boosting Kernel Discriminant Analysis and Its Application on Tissue Classification of Gene Expression Data	Guang Dai, Dit-Yan Yeung
1064	Topological Value Iteration Algorithm for Markov Decision Processes	Peng Dai, Judy Goldsmith

272	Case Base Mining for Adaptation Knowledge Acquisition	Mathieu d.Aquin, Fadi Badra, Sandrine Lafrogne, Jean Lieber, Amedeo Napoli, Laszlo Szathmary
1472	Change of Representation for Statistical Relational Learning	Jesse Davis, Irene Ong, Jan Struyf, Elizabeth Burnside, David Page, V?tor Santos Costa
421	Embedding Non-Ground Logic Programs into Autoepistemic Logic for Knowledge-Base Combination	Jos de Bruijn, Thomas Eiter, Axel Polleres, Hans Tompits
450	Automatic Synthesis of New Behaviors from a Library of Available Behaviors	Giuseppe De Giacomo, Sebastian Sardina
601	Modeling When Connections Are the Problem	Johan de Kleer
641	ProbLog: A Probabilistic Prolog and Its Application in Link Discovery	Luc De Raedt, Angelika Kimmig, Hannu Toivonen
964	Belief Change Based on Global Minimisation	James P. Delgrande, Jérôme Lang, Torsten Schaub
646	A Ranking Approach to Pronoun Resolution	Pascal Denis, Jason Baldridge
1469	Pseudo-Aligned Multilingual Corpora	Fernando Diaz, Donald Metzler
744	Learning Policies for Embodied Virtual Agents through Demonstration	Jonathan Dinerstein, Parris K. Egbert, Dan Ventura
554	Tractable Temporal Reasoning	Clare Dixon, Michael Fisher, Boris Konev
1370	Planning with Goal Utility Dependencies	Minh B. Do, J. Benton, Menkes van den Briel, Subbarao Kambhampati
167	Chronicle Recognition Improvement using Temporal Focusing and Hierarchization	Christophe Dousson, Pierre Le Maigat
1178	Locating Complex Named Entities in Web Text	Doug Downey, Matthew Broadhead, Oren Etzioni
1465	Models of Searching and Browsing: Languages, Studies, and Application	Doug Downey, Susan Dumais, Eric Horvitz

1617	Sharing the Road: Autonomous Vehicles Meet Human Drivers	Kurt Dresner, Peter Stone
708	Word Sense Disambiguation through Sememe Labeling	Xiangyu Duan, Jun Zhao, Bo Xu
675	Learning Classifiers When the Training Data Is Not IID	Murat Dundar, Balaji Krishnapuram, Jinbo Bi, R. Bharat Rao
851	Communicating Effectively in Resource-Constrained Multi-Agent Systems	Partha S. Dutta, Claudia V. Goldman, Nicholas R. Jennings
1075	Kernel Carpentry for Online Regression using Randomly Varying Coefficient Model	Narayanan U. Edakunni, Stefan Schaal, Sethu Vijayakumar
1610	Complexity Results for Checking Equivalence of Stratified Logic Programs	Thomas Eiter, Michael Fink, Hans Tompits, Stefan Woltran
1625	On Reversing Actions: Algorithms and Complexity	Thomas Eiter, Esra Erdem, Wolfgang Faber
951	Case-Based Techniques Used for Dialogue Understanding and Planning in a Human-Robot Dialogue System	Karolina Eliasson
485	An Adaptive Context-based Algorithm for Term Weighting: Application to Single-Word Question Answering	Marco Ernandes, Giovanni Angelini, Marco Gori, Leonardo Rigutini, Franco Scarselli
74	Occam.s Razor Just Got Sharper	Saher Esmeir, Shaul Markovitch
1145	Fault-Model-Based Test Generation for Embedded Software	Michael Esser, Peter Struss
142	Semantic Precision and Recall for Ontology Alignment Evaluation	Jérôme Euzenat
712	The Value of Observation for Monitoring Dynamic Systems	Eyal Even-Dar, Sham M. Kakade, Yishay Mansour
488	Operator Component Matrix Model for IMP Program Diagnosis	Zhao-Fu Fan, Yunfei Jiang
304	On Valued Negation Normal Form Formulas	Hélène Fargier, Pierre Marquis

403	Dealing with Perception Errors in Multi-Robot System Coordination	Alessandro Farinelli, Daniele Nardi, Paul Scerri, Alberto Ingenito
1359	Team Programming in Golog under Partial Observability	Alessandro Farinelli, Alberto Finzi, Thomas Lukasiewicz
1342	Quantified Constraint Satisfaction Problems: From Relaxations to Explanations	Alex Ferguson, Barry O.Sullivan
677	A Decision-Theoretic Model of Assistance	Alan Fern, Sriraam Natarajan, Kshitij Judah, Prasad Tadepalli
956	Sequence Labelling in Structured Domains with Hierarchical Recurrent Neural Networks	Santiago Fernández, Alex Graves, Jürgen Schmidhuber
507	Transferring Learned Control-Knowledge between Planners	Susana Fernández, Ricardo Aler, Daniel Borrajo
1440	A Logic Program Characterization of Causal Theories	Paolo Ferraris
1449	A New Perspective on Stable Models	Paolo Ferraris, Joohyung Lee, Vladimir Lifschitz
1651	WiFi-SLAM Using Gaussian Process Latent Variable Models	Brian Ferris, Dieter Fox, Neil Lawrence
247	Conflict-Based Diagnosis: Adding Uncertainty to Model-based Diagnosis	Ildikó Flesch, Peter Lucas, Theo van der Weide
466	A New Approach for Stereo Matching in Autonomous Mobile Robot Applications	Pasquale Foggia, Jean-Michel Jolion, Alessandro Limongiello, Mario Vento
1488	Voronoi Random Fields: Extracting Topological Structure of Indoor Environments via Place Labeling	Stephen Friedman, Hanna Pasula, Dieter Fox
1193	The Design of ESSENCE: A Constraint Language for Specifying Combinatorial Problems	Alan M. Frisch, Matthew Grum, Chris Jefferson, Bernadette Mart?nez Hern?ndez, Ian Miguel
878	Semi-Supervised Learning for Multi-Component Data Classification	Akinori Fujino, Naonori Ueda, Kazumi Saito

90	Predicting and Preventing Coordination Problems in Cooperative Q-learning Systems	Nancy Fulda, Dan Ventura
1371	Feature Selection and Kernel Design via Linear Programming	Glenn Fung, Romer Rosales, R. Bharat Rao
410	Opinion Sentence Search Engine on Open-domain Blog	Osamu Furuse, Nobuaki Hiroshima, Setsuo Yamada, Ryoji Kataoka
1106	Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis	Evgeniy Gabrilovich, Shaul Markovitch
1241	Learning Restart Strategies	Matteo Gagliolo, Jürgen Schmidhuber
645	Incremental Construction of Structured Hidden Markov Models	Ugo Galassi, Attilio Giordana, Lorenza Saitta
875	On Mining Closed Sets in Multi-Relational Data	Gemma C. Garriga, Roni Khardon, Luc De Raedt
1095	Holonic Multiagent Multilevel Simulation: Application to Real-Time Pedestrian Simulation in Urban Environment	Nicolas Gaud, Franck Gechter, Stéphane Galland, Abderrafiâa Koukam
841	Coordination to Avoid Starvation of Bottleneck Agents in a Large Network System	Rajesh Gautam, Kazuo Miyashita
131	Conflict-Driven Answer Set Solving	Martin Gebser, Benjamin Kaufmann, André Neumann, Torsten Schaub
1203	On Natural Language Processing and Plan Recognition	Christopher W. Geib, Mark Steedman
567	Ranking Alternatives on the Basis of Generic Constraints and Examples -- A Possibilistic Approach	Romain Gérard, Souhila Kaci, Henri Prade
1246	Sellers Competing for Buyers in Online Markets: Reserve Prices, Shill Bids, and Auction Fees	Enrico H. Gerding, Alex Rogers, Rajdeep K. Dash, Nicholas R. Jennings

1643	Improving Embeddings by Flexible Exploitation of Side Information	Ali Ghodsi, Dana Wilkinson, Finnegan Southey
1486	Information-Theoretic Approaches to Branching in Search	Andrew Gilpin, Tuomas Sandholm
673	State Similarity Based Approach for Improving Performance in RL	Sertan Girgin, Faruk Polat, Reda Alhajj
337	Conjunctive Query Answering for the Description Logic SHIQ	Birte Glimm, Ian Horrocks, Carsten Lutz, Uli Sattler
1453	From Sampling to Model Counting	Carla P. Gomes, Joerg Hoffmann, Ashish Sabharwal, Bart Selman
243	ItemRank: A Random-Walk Based Scoring Algorithm for Recommender Engines	Marco Gori, Augusto Pucci
923	Complexity of Pure Equilibria in Bayesian Games	Georg Gottlob, Gianluigi Greco, Toni Mancini
699	Conditional Constraint Satisfaction: Logical Foundations and Complexity	Georg Gottlob, Gianluigi Greco, Toni Mancini
1563	Peripheral-Foveal Vision for Real-time Object Recognition and Tracking in Video	Stephen Gould, Joakim Arfvidsson, Adrian Kaehler, Benjamin Sapp, Marius Messner, Gary Bradski, Paul Baumstarck, Sukwon Chung, Andrew Y. Ng
1340	Generalized Interval Projection: A New Technique for Consistent Domain Extension	Carlos Grandón, Gilles Chabert, Bertrand Neveu
1152	Boosting a Complete Technique to Find MSS and MUS Thanks to a Local Search Oracle	Éric Grégoire, Bertrand Mazure, Cédric Piette
1593	Decidable Reasoning in a Modified Situation Calculus	Yilan Gu, Mikhail Soutchanski
1658	Optimistic Active-Learning using Mutual Information	Yuhong Guo, Russ Greiner

1674	A Dual-Pathway Neural Network Model of Control Relinquishment in Motor Skill Learning	Ashish Gupta, David C. Noelle
1109	Using Ontologies and the Web to Learn Lexical Semantics	Aarti Gupta, Tim Oates
1825	Unsupervised Anomaly Detection	David Guthrie, Louise Guthrie, Ben Allison, Yorick Wilks
1356	Continuous Time Associative Bandit Problems	András György, Levente Kocsis, Ivett Szabó, Csaba Szepesvári
1212	Techniques for Efficient Interactive Configuration of Distribution Networks	Tarik Hadzic, Andrzej Wasowski, Henrik R. Andersen
356	Characterizing Solution Concepts in Games Using Knowledge-Based Programs	Joseph Y. Halpern, Yoram Moses
355	Characterizing the NP-PSPACE Gap in the Satisfiability Problem for Modal Logic	Joseph Y. Halpern, Leandro Chaves Rego
1493	Some Effects of a Reduced Relational Vocabulary on the Whodunit Problem	Daniel T. Halstead, Kenneth D. Forbus
264	Revisiting Output Coding for Sequential Supervised Learning	Guohua Hao, Alan Fern
1639	Maximum Margin Coresets for Active and Noise Tolerant Learning	Sariel Har-Peled, Dan Roth, Dav Zimak
783	Reducing Accidental Complexity in Planning Problems	Patrik Haslum
1176	An Analysis of the Use of Tags in a Blog Recommender System	Conor Hayes, Paolo Avesani, Sriharsha Veeramachaneni
833	Graph-Based Semi-Supervised Learning as a Generative Model	Jingrui He, Jaime Carbonell, Yan Liu
1345	Distance Constraints in Constraint Satisfaction	Emmanuel Hebrard, Barry O. Sullivan, Toby Walsh
313	Hybrid Elections Broaden Complexity-Theoretic Resistance to Control	Edith Hemaspaandra, Lane A. Hemaspaandra, Jörg Rothe

324	Real Boosting a la Carte with an Application to Boosting Oblique Decision Tree	Claudia Henry, Richard Nock, Frank Nielsen
1042	Counting Complexity of Propositional Abduction	Miki Hermann, Reinhard Pichler
1018	Improving LRTA*(k)	Carlos Hernández, Pedro Meseguer
980	Efficient Calculation of Personalized Document Rankings	Claudia Hess, Klaus Stein
856	Planning via Petri Net Unfolding	Sarah Hickmott, Jussi Rintanen, Sylvie Thiébaux, Lang White
1662	Self-Adjusting Ring Modules (SARMs) for Flexible Gait Pattern Generation	Manfred Hild, Frank Pasemann
1766	Analogical Learning in a Turn-Based Strategy Game	Thomas R. Hinrichs, Kenneth D. Forbus
541	Detecting Changes in Unlabeled Data Streams using Martingale	Shen-Shyang Ho, Harry Wechsler
185	SAT Encodings of State-Space Reachability Problems in Numeric Domains	Jörg Hoffmann, Carla Gomes, Bart Selman, Henry Kautz
1308	Truthful Risk-Managed Combinatorial Auctions	Alan Holland, Barry O.Sullivan
32	Adaptation of Organizational Models for Multi-Agent Systems based on Max Flow Networks	Mark Hoogendoorn
1129	Structure Inference for Bayesian Multisensory Perception and Tracking	Timothy M. Hospedales, Joel J. Cartwright, Sethu Vijayakumar
1259	Constraint Partitioning for Solving Planning Problems with TrajectoryConstraints and GoalPreferences	Chih-Wei Hsu, Benjamin W. Wah, Ruoyun Huang, Yixin Chen
400	Collaborative Inductive Logic Programming for Path Planning	Jian Huang, Adrian R. Pearce

686	Constructing New and Better Evaluation Measures for Machine Learning	Jin Huang, Charles X. Ling
406	Extracting Chatbot Knowledge from Online Discussion Forums	Jizhou Huang, Ming Zhou, Dan Yang
219	Observation Reduction for Strong Plans	Wei Huang, Zhonghua Wen, Yunfei Jiang, Lihua Wu
409	The Effect of Restarts on the Efficiency of Clause Learning	Jinbo Huang
1685	An Action Description Language for Iterated Belief Change	Aaron Hunter, James P. Delgrande
1383	Mechanism Design with Partial Revelation	Nathanaël Hyafil, Craig Boutilier
678	Augmented Experiment: Participatory Design with Multiagent Simulation	Toru Ishida, Yuu Nakajima, Yohei Murakami, Hideyuki Nakanishi
1190	Multi-issue Negotiation Protocol for Agents: Exploring Nonlinear Utility Spaces	Takayuki Ito, Hiromitsu Hattori, Mark Klein
248	Using a Hierarchical Bayesian Model to Handle High Cardinality Attributes with Relevant Interactions in a Classification Problem	Jorge Jambeiro Filho, Jacques Wainer
1780	Improving Anytime Point-Based Value Iteration Using Principled Point Selections	Michael R. James, Michael E. Samples, Dmitri A. Dolgov
104	A Three-Stage Neural Model for Attribute Based Classification and Indexing of Fly Ashes	M.A. Jayaram, M.C. Nataraja, C. N. Ravikumar
1485	Dynamic Heuristics for Backtrack Search on Tree-Decomposition of CSPs	Philippe Jégou, Samba Ndojh Ndiaye, Cyril Terrioux
191	A Model for Collective Strategy Diffusion in Agent Social Law Evolution	Yichuan Jiang, Toru Ishida
843	Named Entity Translation with Web Mining and Transliteration	Long Jiang, Ming Zhou, Lee-Feng Chien, Cheng Niu

830	The Role of Macros in Tractable Planning over Causal Graphs	Anders Jonsson
1369	Fast Incremental Square Root Information Smoothing	Michael Kaess, Ananth Ranganathan, Frank Dellaert
1584	Improving Author Coreference by Resource-bounded Information Gathering from the Web	Pallika Kanani, Andrew McCallum, Chris Pal
695	Selective Supervision: Guiding Supervised Learning with Decision-Theoretic Active Learning	Ashish Kapoor, Eric Horvitz, Sumit Basu
1366	Exploiting Sensorimotor Coordination for Learning to Recognize Objects	Yohannes Kassahun, Mark Edgington, Jose de Gea, Frank Kirchner
1112	Factored Planning using Decomposition Trees	Elena Kelareva, Olivier Buffet, Jinbo Huang, Sylvie Thiébaux
80	Property Persistence in the Situation Calculus	Ryan F. Kelly, Adrian R. Pearce
284	Computation of Initial Modes for K-modes Clustering Algorithm using Evidence Accumulation	Shehroz S. Khan, Shri Kant
635	Symmetric Component Caching	Matthew Kitching, Fahiem Bacchus
590	Sequential Bundle-Bid Single-Sale Auction Algorithms for Decentralized Control	Sven Koenig, Craig Tovey, Xiaoming Zheng, Ilgaz Sungur
1705	Optimizing Classifier Performance in Word Sense Disambiguation by Redefining Sense Classes	Upali S. Kohomban, Wee Sun Lee
750	Avoidance of Model Re-Induction in SVM-based Feature Selection for Text Categorization	Aleksander Kolcz, Abdur Chowdhury
1210	Building Portable Options: Skill Transfer in Reinforcement Learning	George Konidaris, Andrew Barto

140	Recent Progress in Heuristic Search: A Case Study of the Four-Peg Towers of Hanoi Problem	Richard E. Korf, Ariel Felner
845	A Factor Graph Model for Software Bug Finding	Ted Kremenek, Andrew Y. Ng, Dawson Engler
99	Logistic Regression Models for a Fast CBIR Method Based on Feature Selection	Riadh Ksantini, Djemel Ziou, Bernard Colin, Francois Dubeau
663	Fast (Incremental) Algorithms for Useful Classes of Simple Temporal Problems with Preferences	T. K. Satish Kumar
991	Learning from the Report-writing Behavior of Individuals	Mohit Kumar, Nikesh Garera, Alexander I. Rudnicky
163	Optimal Multi-Sensor based Multi Target Detection by Moving Sensors to the Maximal Clique in a Covering Graph	Ganesh P. Kumar, K. Madhava Krishna
242	Collapsed Variational Dirichlet Process Mixture Models	Kenichi Kurihara, Max Welling, Yee Whye Teh
449	Handling Alternative Activities in Resource-Constrained Project Scheduling Problems	J?rgen Kuster, Jannach Dietmar, Gerhard Friedrich
753	Marginalized Multi-Instance Kernels	James T. Kwok, Pak-Ming Cheung
205	SegGen: A Genetic Algorithm for Linear Text Segmentation	Sylvain Lamprier, Tassadit Amghar, Bernard Levrat, Frederic Saubion
915	r-grams: Relational Grams	Niels Landwehr, Luc De Raedt
894	Belief Update Revisited	Jérôme Lang
948	Vote and Aggregation in Combinatorial Domains with Structured Preferences	Jérôme Lang
1110	Winner Determination in Sequential Majority Voting	Jérôme Lang, Maria Silvia Pini, Francesca Rossi, K. Brent Venable, Toby Walsh
792	Adaptive Genetic Algorithm with Mutation and Crossover Matrices	Nga Lam Law, Kwok Yip Szeto

904	A Study of Residual Supports in Arc Consistency	Christophe Lecoutre, Fred Hemery
898	Nogood Recording from Restarts	Christophe Lecoutre, Lakhdar Sais, S?bastien Tabary, Vincent Vidal
1153	Dynamically Weighted Hidden Markov Model for Spam Deobfuscation	Seunghak Lee, Iryoung Jeong, Seungjin Choi
693	RoxyBot-06: An (SAA)2 TAC Travel Agent	Seong Jae Lee, Amy Greenwald, Victor Naroditskiy
566	Combining Topological and Directional Information for Spatial Reasoning	Sanjiang Li
891	Generalized Additive Bayesian Network Classifiers	Jianguo Li, Changshui Zhang, Tao Wang, Yimin Zhang
1100	Generalizing the Bias Term of Support Vector Machines	Wenye Li, Kwong-Sak Leung, Kin-Hong Lee
381	Learning to Identify Unexpected Instances in the Test Set	Xiao-li Li, Bing Liu, See-Kiong Ng
114	Robust Object Tracking with a Case-base Updating Strategy	Wenhui Liao, Yan Tong, Zhiwei Zhu, Qiang Ji
1735	Training Conditional Random Fields using Virtual Evidence Boosting	Lin Liao, Tanzeem Choudhury, Dieter Fox, Henry Kautz
1141	Arc Consistency during Search	Chavalit Likitvivatanavong, Yuanlin Zhang, Scott Shannon, James Bowen, Eugene C. Freuder
1664	Explanation-Based Feature Construction	Shiau Hong Lim, Li-Lun Wang, Gerald DeJong
1118	Exploiting Inference Rules to Compute Lower Bounds for MAX-SAT Solving	Han Lin, Kaile Su
1170	From Answer Set Logic Programming to Circumscription via Logic of GK	Fangzhen Lin, Yi Zhou
714	Cluster-Based Selection of Statistical Answering Strategies	Lucian Vlad Lita, Jaime Carbonell
718	Bayesian Tensor Inference for Sketch-Based Facial Photo Hallucination	Wei Liu, Xiaoou Tang, Jianzhuang Liu

749	Feature Mining and Neuro-Fuzzy Inference System for Steganalysis of LSB Matching Stegangoraphy in Grayscale Images	Qingzhong Liu, Andrew H. Sung
739	Protein Quaternary Fold Recognition Using Conditional Graphical Models	Yan Liu, Jaime Carbonell, Vanathi Gopalakrishnan, Peter Weigele
1295	Automatic Gait Optimization with Gaussian Process Regression	Daniel Lizotte, Tao Wang, Michael Bowling, Dale Schuurmans
461	Automatic Verification of Knowledge and Time with NuSMV	Alessio Lomuscio, Charles Pecheur, Franco Raimondi
1513	Incremental Learning of Perceptual Categories for Open-Domain Sketch Recognition	Andrew Lovett, Morteza Dehghani, Kenneth Forbus
1777	Recursive Random Fields	Daniel Lowd, Pedro Domingos
282	Prediction of Probability of Survival in Critically Ill Patients Optimizing the Area under the ROC Curve	Oscar Luaces, José R. Quevedo, Francisco Taboada, Guillermo M. Albaiceta, Antonio Bahamonde
866	Conservative Extensions in Expressive Description Logics	Carsten Lutz, Dirk Walther, Frank Wolter
204	An Energy-Efficient, Multi-Agent Sensor Network for Detecting Diffuse Events	Rónán Mac Ruairí, Mark T. Keane
1290	Towards a Computational Model of Melody Identification in Polyphonic Music	Søren Tjagvad Madsen, Gerhard Widmer
208	A Multi-agent Medical System for Indian Rural Infant and Child Care	Vijay Kumar Mago, M. Syamala Devi
771	Automatically Selecting Answer Templates to Respond to Customer Emails	Rahul Malik, L. Venkata Subramaniam, Saroj Kaushik
1326	Infeasibility Certificates and the Complexity of the Core in Coalitional Games	Enrico Malizia, Luigi Palopoli, Francesco Scarcello

630	A Multiobjective Frontier Search Algorithm	Lawrence Mandow, José Luis Pérez de la Cruz
420	Providing a Recommended Trading Agent to a Population: A Novel Approach	Efrat Manistersi, Ron Katz, Sarit Kraus
426	Enhancing MAS Cooperative Search Through Coalition Partitioning	Efrat Manisterski, David Sarne, Sarit Kraus
342	A Fast Analytical Algorithm for Solving Markov Decision Processes with Real-Valued Resources	Janusz Marecki, Sven Koenig, Milind Tambe
123	Topological Mapping through Distributed, Passive Sensors	Dimitri Marinakis, Gregory Dudek
1680	A Predictive Approach to Help-Desk Response Generation	Yuval Marom, Ingrid Zukerman
1432	Modelling Well-Structured Argumentation Lines	Diego C. Martínez, Alejandro J. García, Guillermo R. Simari
1243	A Distributed Architecture for Symbolic Data Fusion	Fulvio Mastrogiovanni, Antonio Sgorbissa, Renato Zaccaria
1568	A Comparison of Time-Space Schemes for Graphical Models	Robert Mateescu, Rina Dechter
1555	Inferring Long-term User Properties Based on Users. Location History	Yutaka Matsuo, Naoaki Okazaki, Kiyoshi Izumi, Yoshiyuki Nakamura, Takuichi Nishimura, K?iti Hasida, Hideyuki Nakashima
1414	Efficient HPSG Parsing with Supertagging and CFG-Filtering	Takuya Matsuzaki, Yusuke Miyao, Jun.ichi Tsujii
271	Planning for Temporally Extended Goals as Propositional Satisfiability	Robert Mattmüller, Jussi Rintanen
1532	A Hybridized Planner for Stochastic Domains	Mausam, Piergiorgio Bertoli, Daniel S. Weld
1402	Abstract Interpretation of Programs for Model-based Debugging	Wolfgang Mayer, Markus Stumptner
343	The Ins and Outs of Critiquing	David McSherry, David W. Aha

1754	A Flexible Unsupervised PP-Attachment Method Using Semantic Information	Srinivas Medimi, Pushpak Bhattacharyya
615	Probabilistic Consistency Boosts MAC and SAC	Deepak Mehta, M. R. C. van Dongen
1299	Performance Analysis of Online Anticipatory Algorithms for Large Multistage Stochastic Integer Programs	Luc Mercier, Pascal Van Hentenryck
702	Hierarchical Heuristic Forward Search in Stochastic Domains	Nicolas Meuleau, Ronen I. Brafman
253	Learning from Partial Observations	Loizos Michael
1148	Compiling Bayesian Networks by Symbolic Probability Calculation Based on Zero-suppressed BDDs	Shin-ichi Minato, Ken Satoh, Taisuke Sato
1527	Improving Activity Discovery with Automatic Neighborhood Estimation	David Minnen, Thad Starner, Irfan Essa, Charles Isbell
240	Generalizing Temporal Controllability	Michael D. Moffitt, Martha E. Pollack
774	Multipotential Games	Dov Monderer
1737	An Extension to Conformant Planning using Logic Programming	A. Ricardo Morales, Phan Huy Tu, Tran Cao Son
1816	Extracting Keyphrases to Represent Relations in Social Networks from Web	Junichiro Mori, Ishizuka Mitsuru, Yutaka Matsuo
325	A Faithful Integration of Description Logics with Logic Programming	Boris Motik, Riccardo Rosati
1503	Evaluating a Decision-Theoretic Approach to Tailored Example Selection	Kasia Muldner, Cristina Conati
57	Expectation Failure as a Basis for Agent-Based Model Diagnosis and Mixed Initiative Model Adaptation during Anomalous Plan Execution	Alice Mulvehill, Brett Benyo, Michael Cox, Renu Bostwick
1034	Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters	Volker Nannen, A.E. Eiben

1005	Local Search for Balanced Submodular Clusterings	Mukund Narasimhan, Jeff Bilmes
398	Efficiently Exploiting Symmetries in Real Time Dynamic Programming	Shravan Matthur Narayanamurthy, Balaraman Ravindran
231	Constraint and Variable Ordering Heuristics for Compiling Configuration Problems	Nina Narodytska, Toby Walsh
704	Hierarchical Multi-channel Hidden Semi Markov Models	Pradeep Natarajan, Ramakant Nevatia
1142	Consistency Checking of Basic Cardinal Constraints over Connected Regions	Isabel Navarrete, Antonio Morales, Guido Sciavicco
880	Graph Connectivity Measures for Unsupervised Word Sense Disambiguation	Roberto Navigli, Mirella Lapata
270	Iterated Belief Contraction from First Principles	Abhaya C. Nayak, Randy Goebel, Mehmet A. Orgun
1758	Shallow Semantics for Coreference Resolution	Vincent Ng
818	Kernel Matrix Evaluation	Canh Hao Nguyen, Tu Bao Ho
786	Subtree Mining for Question Classification Problem	Minh Le Nguyen, Thanh Tri Nguyen, Akira Shimazu
1394	A Theoretical Framework for Learning Bayesian Networks with Parameter Inequality Constraints	Radu Stefan Niculescu, Tom M. Mitchell, R. Bharat Rao
649	Neighborhood MinMax Projections	Feiping Nie, Shiming Xiang, Changshui Zhang
1687	Extracting and Visualizing Trust Relationships from Online Auction Feedback Comments	John O.Donovan, Barry Smyth, Vesile Evrim, Dennis McLeod
1136	Multi-Agent System that Attains Longevity via Death	Megan Olsen, Hava Siegelmann
815	Case-based Learning from Proactive Communication	Santi Ontañón, Enric Plaza

572	Argumentation Based Contract Monitoring in Uncertain Domains	Nir Oren, Timothy J. Norman, Alun Preece
1192	Learning to Count by Think Aloud Imitation	Laurent Orseau
1088	Dynamic Verification of Trust in Distributed Open Systems	Nardine Osman, David Robertson
331	Co-Localization from Labeled and Unlabeled Data Using Graph Laplacian	Jeffrey Junfeng Pan, Qiang Yang
1124	Natural Language Query Recommendation in Conversation Systems	Shimei Pan, James Shaw
164	Feature Based Occupancy Grid Maps for Sonar Based Safe-Mapping	Amit Kumar Pandey, K. Madhava Krishna, Mainak Nath
944	Probabilistic Go Theories	Austin Parker, Fusun Yaman, Dana Nau, V.S. Subrahmanian
1761	What You Seek is What You Get: Extraction of Class Attributes from Query Logs	Marius Pasca, Benjamin Van Durme
976	Grounding for Model Expansion in k-Guarded Formulas with Inductive Definitions	Murray Patterson, Yongmei Liu, Eugenia Ternovska, Arvind Gupta
698	Quality Guarantees on k-Optimal Solutions for Distributed Constraint Optimization Problems	Jonathan P. Pearce, Milind Tambe
1353	State Space Search for Risk-Averse Agents	Patrice Perny, Olivier Spanjaard, Louis-Xavier Storme
1043	MB-DPOP: A New Memory-Bounded Algorithm for Distributed Optimization	Adrian Petcu, Boi Faltings
862	PC-DPOP: A New Partial Centralization Algorithm for Distributed Optimization	Adrian Petcu, Boi Faltings, Roger Mailler
1562	An Analysis of Laplacian Methods for Value Function Approximation in MDPs	Marek Petrik
1195	Average-Reward Decentralized Markov Decision Processes	Marek Petrik, Shlomo Zilberstein

143	Probabilistic Mobile Manipulation in Dynamic Environments, with Application to Opening Doors	Anna Petrovskaya, Andrew Y. Ng
445	DiPRA: Distributed Practical Reasoning Architecture	Giovanni Pezzulo, Gianguglielmo Calvi, Cristiano Castelfranchi
174	Global/Local Dynamic Models	Avi Pfeffer, Subrata Das, David Lawless, Brenda Ng
839	Building Structure into Local Search for SAT	Duc Nghia Pham, John Thornton, Abdul Sattar
527	A Tighter Error Bound for Decision Tree Learning Using PAC Learnability	Chaithanya Pichuka, Raju S. Bapi, Chakravarthy Bhagvati, Arun K. Pujari, Bulusu L. Deekshatulu
1090	Incompleteness and Incomparability in Preference Aggregation	Maria Silvia Pini, Francesca Rossi, K. Brent Venable, Toby Walsh
472	Surprise as Shortcut for Anticipation: Clustering Mental States in Reasoning	Michele Piunti, Cristiano Castelfranchi, Rino Falcone
1682	Efficient Failure Detection on Mobile Robots Using Particle Filters with Gaussian Process Proposals	Christian Plagemann, Dieter Fox, Wolfram Burgard
178	Semi-Supervised Learning of Attribute-Value Pairs from Product Descriptions	Katharina Probst, Rayid Ghani, Marko Krema, Andrew Fano, Yan Liu
221	Gossip-Based Aggregation of Trust in Decentralized Reputation Systems	Ariel D. Procaccia, Yoram Bachrach, Jeffrey S. Rosenschein
161	Multi-Winner Elections: Complexity of Manipulation, Control and Winner-Determination	Ariel D. Procaccia, Jeffrey S. Rosenschein, Aviv Zohar
967	Automated Benchmark Model Generators for Model-Based Diagnostic Inference	Gregory Provan, Jun Wang

1411	Automated Heart Wall Motion Abnormality Detection from Ultrasound Images using Bayesian Networks	Maleeha Qazi, Glenn Fung, Sriram Krishnan, Romer Rosales, Harald Steck, R. Bharat Rao, Don Poldermans, Dhanalakshmi Chandrasekaran
1334	Near-Optimal Anytime Coalition Structure Generation	Talal Rahwan, Sarvapali D. Ramchurn, Viet Dung Dang, Nicholas R. Jennings
1739	Bayesian Inverse Reinforcement Learning	Deepak Ramachandran, Eyal Amir
1361	Loopy SAM	Ananth Ranganathan, Michael Kaess, Frank Dellaert
1542	Kernel Conjugate Gradient for Fast Kernel Machines	Nathan D. Ratliff, J. Andrew Bagnell
287	Deictic Option Schemas	Balaraman Ravindran, Andrew G. Barto, Vimal Mathew
569	Real-Time Heuristic Search with a Priority Queue	D. Chris Rayner, Katherine Davison, Vadim Bulitko, Kenneth Anderson, Jieshan Lu
1810	Qualitative Spatial and Temporal Reasoning: Efficient Algorithms for Everyone	Jochen Renz
1667	Opponent Modeling in Scrabble	Mark Richards, Eyal Amir
697	Diagnosability Testing with Satisfiability Algorithms	Jussi Rintanen, Alban Grastien
690	Diagnosers and Diagnosability of Succinct Transition Systems	Jussi Rintanen
537	A fusion of Stacking with Dynamic Integration	Niall Rooney, David Patterson
933	Acquiring a Robust Case Base for the Robot Soccer Domain	Raquel Ros, Josep Lluís Arcos
1385	AEMS: An Anytime Online Search Algorithm for Approximate Policy Refinement in Large POMDPs	Stéphane Ross, Brahim Chaib-draa

574	Efficient Bayesian Task-Level Transfer Learning	Daniel M. Roy, Leslie P. Kaelbling
279	Routing Mediators	Ola Rozenfeld, Moshe Tennenholtz
1630	Best-first Utility-guided Search	Wheeler Ruml, Minh B. Do
377	Graph Decomposition for Efficient Multi-Robot Path Planning	Malcolm Ryan
768	Robust Human-Computer Interaction System Guiding a User by Providing Feedback	Michael S. Ryoo, Jake K. Aggarwal
1339	An efficient protocol for negotiation over multiple indivisible resources	Sabyasachi Saha, Sandip Sen
659	QuantMiner: A Genetic Algorithm for Mining Quantitative Association Rules	Ansaf Salleb-Aouissi, Christel Vrain, Cyril Nortet
1085	Automated Design of Multistage Mechanisms	Tuomas Sandholm, Vincent Conitzer, Craig Boutilier
1560	A Dynamic Approach for MPE and Weighted MAX-SAT	Tian Sang, Paul Beame, Henry Kautz
189	Inside-Outside Probability Computation for Belief Propagation	Taisuke Sato
1603	Is the Turing Test Good Enough? The Fallacy of Resource-Unbounded Intelligence	Virginia Savova, Leonid Peshkin
250	Depth Estimation using Monocular and Stereo Cues	Ashutosh Saxena, Jamie Schulte, Andrew Y. Ng
176	OSS: A Semantic Similarity Function based on Hierarchical Ontologies	Vincent Schickel-Zuber, Boi Faltings
215	Description Logics with Approximate Definitions . Precise Modeling of Vague Concepts	Stefan Schlobach, Michel Klein, Linda Peelen
682	A Size-Based Qualitative Approach to the Representation of Spatial Granularity	Hedda R. Schmidtke, Woontack Woo
938	Qualitative Temporal Reasoning about Vague Events	Steven Schockaert, Martine De Cock, Etienne E. Kerre

1599	Scalable Diagnosability Checking of Event-Driven Systems	Anika Schumann, Yannick Pencol?
1215	Combining Learning and Word Sense Disambiguation for Intelligent User Profiling	Giovanni Semeraro, Marco Degemmis, Pasquale Lops, Pierpaolo Basile
1572	Emergence of Norms through Social Learning	Sandip Sen, Stéphane Airiau
1696	Memory-Bounded Dynamic Programming for DEC-POMDPs	Sven Seuken, Shlomo Zilberstein
1675	Logical Circuit Filtering	Dafna Shahaf, Eyal Amir
42	Forward Search Value Iteration for POMDPs	Guy Shani, Ronen I. Brafman, Solomon E. Shimony
1080	Dynamic Interactions Between Goals and Beliefs	Steven Shapiro, Gerhard Brewka
1116	Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL	Manu Sharma, Michael Holmes, Juan Santamaria, Arya Irani, Charles Isbell, Ashwin Ram
1805	Document Summarization using Conditional Random Fields	Dou Shen, Jian-Tao Sun, Hua Li, Qiang Yang, Zheng Chen
1519	Real-Time Detection of Task Switches of Desktop Users	Jianqiang Shen, Lida Li, Thomas G. Dietterich
1533	A Dual-layer CRFs Based Joint Decoding Method for Cascaded Segmentation and Labeling Tasks	Yanxin Shi, Mengqiu Wang
82	Parametric Kernels for Sequence Data Analysis	Young-In Shin, Donald Fussell
689	Hierarchical Diagnosis of Multiple Faults	Sajjad Siddiqi, Jinbo Huang
305	Information-Based Agency	Carles Sierra, John Debenham
605	Reinforcement Learning of Local Shape in the Game of Go	David Silver, Richard Sutton, Martin M?ller
352	Semi-Supervised Gaussian Process Classifiers	Vikas Sindhwani, Wei Chu, S. Sathiya Keerthi

489	Detection of Cognitive States from fMRI Data using Machine Learning Techniques	Vishwajeet Singh, Krishna P. Miyapuram, Raju S. Bapi
1611	Efficient Planning of Informative Paths for Multiple Robots	Amarjeet Singh, Andreas Krause, Carlos Guestrin, William Kaiser, Maxim Batalin
914	Estimating the rate of Web Page Updates	Sanasam Ranbir Singh
235	Formalizing Communication Protocols for Multiagent Systems	Munindar P. Singh
794	Control of Agent Swarms using Generalized Centroidal Cyclic Pursuit Laws	Arpita Sinha, Debasish Ghose
1640	Database-Text Alignment via Structured Multilabel Classification	Benjamin Snyder, Regina Barzilay
1645	Inferring Complex Agent Motions from Partial Trajectory Observations	Finnegan Southey, Wesley Loh, Dana Wilkinson
478	Color Learning on a Mobile Robot: Towards Full Autonomy under Changing Illumination	Mohan Sridharan, Peter Stone
246	On the Automatic Scoring of Handwritten Essays	Sargur Srihari, Rohini Srihari, Pavithra Babu, Harish Srinivasan
596	Domain Independent Approaches for Finding Diverse Plans	Biplav Srivastava, Tuan A. Nguyen, Alfonso Gerevini, Subbarao Kambhampati, Minh Binh Do, Ivan Serina
124	Online Speed Adaptation using Supervised Learning for High-Speed, Off-Road Autonomous Driving	David Stavens, Gabriel Hoffmann, Sebastian Thrun
---	Learning and Multiagent Reasoning for Autonomous Agents	Peter Stone
919	Dances with Words	Carlo Strapparava, Alessandro Valitutti, Oliviero Stock
1138	Model-based Optimization of Testing through Reduction of Stimuli	Peter Struss

134	Backtracking Procedures for Hypertree, HyperSpread and Connected Hypertree Decomposition of CSPs	Sathiamoorthy Subbarayan, Henrik Reif Andersen
913	A General Framework for Reasoning about Inconsistency	V.S. Subrahmanian, Leïla Amgoud
1426	On Modeling Multiagent Task Scheduling as a Distributed Constraint Optimization Problem	Evan A. Sultanik, Pragnesh Jay Modi, William C. Regli
58	Dynamic Weighting A* Search-based MAP Algorithm for Bayesian Networks	Xiaoxun Sun, Marek J. Druzdzel, Changhe Yuan
598	The Fringe-Saving A* Search Algorithm . A Feasibility Study	Xiaoxun Sun, Sven Koenig
888	Appearance based Recognition Methodology for Recognising Fingerspelling Alphabets	M.G. Suraj, D.S. Guru
1262	An Experts Algorithm for Transfer Learning	Erik Talvitie, Satinder Singh
864	Direct Code Access in Self-Organizing Neural Networks for Reinforcement Learning	Ah-Hwee Tan
1011	Layout Analysis of Tree-Structured Scene Frames in Comic Images	Takamasa Tanaka, Kenji Shoji, Fubito Toyama, Juichi Miyamichi
1250	Emotions as Durative Dynamic State for Action Selection	Emmanuel Tanguy, Philip Willis, Joanna J. Bryson
738	Grounding Abstractions in Predictive State Representations	Brian Tanner, Vadim Bulitko, Anna Koop, Cosmin Paduraru
1111	Metric Properties of Structured Data Visualizations through Generative Probabilistic Modeling	Peter Tino, Nikolaos Gianniotis
380	Face Recognition via the Overlapping Energy Histogram	Ronny Tjahyadi, Wanquan Liu, Senjian An, Svetha Venkatesh
1827	Planning under Risk and Knightian Uncertainty	Felipe W. Trevizan, Fábio G. Cozman, Leliane N. de Barros

1225	Instace-based AMN Classification for Improved Object Recognition in 2D and 3D Laser Range Data	Rudolph Triebel, Richard Schmidt, óscar Martínez Mozos, Wolfram Burgardd
391	Ensembles of Partially Trained SVMs with Multiplicative Updates	Ivor W. Tsang, James T. Kwok
885	Word Sense Disambiguation with Spreading Activation Networks Generated from Thesauri	George Tsatsaronis, Michalis Vazirgiannis, Ion Androutsopoulos
1213	Morphological Annotation of a Large Spontaneous Speech Corpus in Japanese	Kiyotaka Uchimoto, Hitoshi Isahara
997	Speaker-Invariant Features for Automatic Speech Recognition	Srinivasan Umesh, D. Rama Sanand, G. Praveen
1062	Correlation Clustering for Crosslingual Link Detection	Jurgen Van Gael, Xiaojin Zhu
198	Towards Efficient Computation of Error Bounded Solutions in POMDPs: Expected Value Approximation and Dynamic Disjunctive Beliefs	Pradeep Varakantham, Rajiv T. Maheswaran, Tapana Gupta, Milind Tambe
1752	Resource Constraints on Computation and Communication in the Brain	Sashank Varma
722	Progression of Situation Calculus Action Theories with Incomplete Information	Stavros Vassos, Hector Levesque
835	Semantic Indexing of a Competence Map to Support Scientific Collaboration in a Research Community	Paola Velardi, Roberto Navigli, Michaël Petit
1285	An Experience on Reputation Models Interoperability Based on a Functional Ontology	Laurent Vercouter, Sara J. Casare, Jaime S. Sichman, Anarosa A. F. Brand?o
858	A labeling approach to the computation of credulous acceptance in argumentation	Bart Verheij
1388	Generating Bayes-Nash Equilibria to Design Autonomous Trading Agents	Ioannis A. Vetsikas, Nicholas R. Jennings, Bart Selman

127	MESH-Based Active Monte Carlo Recognition (MESH-AMCR)	Felix von Hundelshausen, H. J. Wuensche, Marco Block, Raul Kompass, Raúl Rojas
115	Manifold-Ranking Based Topic-Focused Multi-Document Summarization	Xiaojun Wan, Jianwu Yang, Jianguo Xiao
41	A Convengent Solution to Tensor Subspace Learning	Huan Wang, Shuicheng Yan, Thomas Huang, Xiaoou Tang
511	A Hybrid Ontology Directed Feedback Selection Algorithm for Supporting Creative Problem Solving Dialogues	Hao-Chuan Wang, Rohit Kumar, Carolyn Penstein Rosé, Tsai-Yen Li, Chun-Yen Chang
987	All Common Subsequences	Hui Wang
1672	Common Sense Based Joint Training of Human Activity Recognizers	Shiaokai Wang, William Pentney, Ana-Maria Popescu, Tanzeem Choudhury, Matthai Philipose
573	First Order Decision Diagrams for Relational MDPs	Chenggang Wang, Saket Joshi, Roni Khardon
1184	Formal Trust Model for Multiagent Systems	Yonghong Wang, Munindar P. Singh
1156	Self-Adaptive Neural Networks Based on a Poisson Approach for Knowledge Discovery	Haiying Wang, Huiru Zheng, Francisco Azuaje
1701	Simple Training of Dependency Parsers via Structured Boosting	Qin Iris Wang, Dekang Lin, Dale Schuurmans
727	Dynamic Mixture Models for Multiple Time-Series	Xing Wei, Jimeng Sun, Xuerui Wang
107	Using Graph Algebra to Optimize Neighborhood for Isometric Mapping	Guihua Wen, Lijun Jiang, Nigel R. Shadbolt
517	Dynamics of Temporal Difference Learning	Andreas Wendemuth
1081	Machine Learning for On-Line Hardware Reconfiguration	Jonathan Wildstrom, Peter Stone, Emmett Witchel, Mike Dahlin

1216	A Primitive Based Generative Model to Infer Timing Information in Unpartitioned Handwriting Data	Ben H. Williams, Marc Toussaint, Amos J. Storkey
540	Relational Knowledge with Predictive State Representations	David Wingate, Vishal Soni, Britton Wolfe, Satinder Singh
1678	One Class per Named Entity: Exploiting Unlabeled Text for Named Entity Recognition	Yingchuan Wong, Hwee Tou Ng
733	Using a Mobile Robot for Cognitive Mapping	Chee K. Wong, Jochen Schmidt, Wai K. Yeap
1253	Representations for Action Selection Learning from Real-Time Observation of Task Experts	Mark A. Wood, Joanna J. Bryson
319	A Subspace Kernel for Nonlinear Feature Extraction	Mingrui Wu, Jason Farquhar
1722	A Privacy-Sensitive Approach to Modeling Multi-Person Conversations	Danny Wyatt, Tanzeem Choudhury, Jeff Bilmes, Henry Kautz
834	Discriminative Learning of Beam-Search Heuristics for Planning	Yuehua Xu, Alan Fern, Sungwook Yoon
1392	Understanding Drawings by Compositional Analogy	Patrick W. Yaner, Ashok K. Goel
867	A Call Admission Control Scheme using NeuroEvolution Algorithm in Cellular Networks	Xu Yang, John Bigham
252	A Scalable Kernel-Based Algorithm for Semi-Supervised Metric Learning	Dit-Yan Yeung, Hong Chang, Guang Dai
351	Multi-Document Summarization by Maximizing Informative Content-Words	Wen-tau Yih, Joshua Goodman, Lucy Vanderwende, Hisami Suzuki
122	Automatic Decision of Piano Fingering Based on a Hidden Markov Models	Yuichiro Yonebayashi, Hirokazu Kameoka, Shigeki Sagayama
724	Using Learned Policies in Heuristic-Search Planning	SungWook Yoon, Alan Fern, Robert Givan

1430	Ambiguous Part-of-Speech Tagging for Improving Accuracy and Domain Portability of syntactic parsers	Kazuhiro Yoshida, Yoshimasa Tsuruoka, Yusuke Miyao, Jun. ichi Tsujii
1375	Lambda Depth-first Proof Number Search and its Application to Go	Kazuki Yoshizoe, Akihiro Kishimoto, Martin Müller
186	Mean Shift as Half Quadratic Optimization: With Application to Sequential Mode Seeking	XiaoTong Yuan
1026	Managing Domain Knowledge and Multiple Models with Boosting	Peng Zang, Charles Isbell
260	Towards Runtime Behavior Adaptation for Embodied Characters	Peng Zang, Manish Mehta, Michael Mateas, Ashwin Ram
1097	Concept Sampling: Towards Systematic Selection in Large-Scale Mixed Concepts in Machine Learning	Yi Zhang, Xiaoming Jin
262	Epistemic Reasoning in Logic Programs	Yan Zhang
746	Fast Algorithm for Connected Row Convex Constraints	Yuanlin Zhang
824	Automatic Acquisition of Context-Specific Lexical Paraphrases	Shiqi Zhao, Ting Liu, Xincheng Yuan, Sheng Li, Yu Zhang
820	Learning Question Paraphrases for QA from Encarta Logs	Shiqi Zhao, Ming Zhou, Ting Liu
883	Searching for Interacting Features	Zheng Zhao, Huan Liu
1666	Edge Partitioning in External-Memory Graph Search	Rong Zhou, Eric A. Hansen
897	Exploiting Image Contents in Web Search	Zhi-Hua Zhou, Hong-Bin Dai
1504	Learning User Clicks in Web Search	Ding Zhou, Levent Bolelli, Jia Li, C. Lee Giles, Hongyuan Zha
1753	Semantic Smoothing of Document Models for Agglomerative Clustering	Xiaohua Zhou, Xiaodan Zhang, Xiaohua Hu
1547	An Empirical Study of the Noise Impact on Cost-Sensitive Learning	Xingquan Zhu, Xindong Wu, Taghi M. Khoshgoftaar, Yong Shi

591	Mining Complex Patterns across Sequences with Gap Requirements	Xingquan Zhu, Xindong Wu
386	Conflict Directed Backjumping for Max-CSPs	Roie Zivan, Amnon Meisels
397	Using Focal Point Learning to Improve Tactic Coordination in Human-Machine Interactions	Inon Zuckerman, Sarit Kraus, Jeffrey S. Rosenschein

Workshop on Partial Evaluation and Program Manipulation

Nice, France, January 15-16, 2007

http://www.program-transformation.org/PEPM07

Accepted Papers

- Ralf Laemmel. *Style normalization for canonical X-to-O mappings*
- Dongxi Liu, Zhenjiang Hu and Masato Takeichi. *Bidirectional Interpretation of XQuery*
- Alcino Cunha and Joost Visser. *Transformation of Structure-Shy Programs -- Applied to XPath Queries and Strategic Functions*
- Jacques Carette and Michael Kucera. *Partial Evaluation for Maple*
- German Vidal. *Quasi-Terminating Logic Programs for Ensuring the Termination of Partial Evaluation*
- Geoff Hamilton. *Distillation: Extracting the Essence of Programs*
- Kimberley Burchett, Gregory Cooper and Shriram Krishnamurthi. *Lowering: A Static Optimization Technique for Functional Reactive Languages*
- Tom Rothamel and Annie Liu. *Efficient Implementation of Tuple Pattern Retrieval*
- Coen De Roover, Johan Brichau, Carlos Noguera, Theo D'Hondt and Laurence Duchien. *Behavioral Similarity Matching using Concrete Source Code Templates in Logic Queries*
- João Fernandes and João Saraiva. *Tools and Libraries to Model and Manipulate Circular Programs*
- Emir Pasalic, Jeremy Siek, Walid Taha and Seth Fogarty. *Concoqtion: Indexed Types Now!*
- Walid Taha and Stephan Elnner. *The Semantics of Graphical Languages*

- Jeffrey Fischer, Rupak Majumdar and Todd Millstein. *Tasks: Language Support for Event-driven Programming*
- Tetsuo Yokoyama and Robert Glueck. *A Reversible Programming Language and its Invertible Self-Interpreter*
- Ping Zhu and Siau-Cheng Khoo. *Towards Constructing Reusable Specialization Components*
- Claudio Ochoa and German Puebla. *Poly-Controlled Partial Evaluation In Practice*

Net Talk

edited by Roberto Bagnara

Content:

Good Examples of Properly Commented Prolog Code

From: **Roberto Bagnara**

Can anyone point me to (very) good examples of properly commented Prolog code? I am looking for coding and commenting styles and practices upon which to base the development of rather big applications.

From: **Steve Moyle**
Subject: Re: Good Examples of Properly Commented Prolog Code

One place to start would be Covington's guidlines:

[PDF] Some Coding Guidelines for Prolog
http://www.ai.uga.edu/mc/plcoding.pdf

I am interested how this thread develops, particularly those that have experience of professional development in Prolog.

From: **Paulo Moura**
Subject: Re: Good Examples of Properly Commented Prolog Code

Roberto Bagnara wrote:

```
> can anyone point me to (very) good examples of properly
commented
> Prolog code?  I am looking for coding and commenting
styles and
> practices upon which to base the development of rather big
> applications.
```

Maybe the following page will give you some ideas:

http://www.logtalk.org/contributions/iso8601.html

From: **Richard A. O'Keefe**
Subject: Re: Good Examples of Properly Commented Prolog Code

Steve Moyle wrote:

```
> One place to start would be Covington's guidlines:
>
> [PDF] Some Coding Guidelines for Prolog
> http://www.ai.uga.edu/mc/plcoding.pdf
```

I have a few quibbles with that.

1.1 Begin every predicate (except auxiliary predicates) with an introductory comment in the standard format.
Quibble: There is no "THE" standard format. The form I use begins with

```
%   <name>(<mode><var>: <type>, ..., <mode><var>:
<type>)
%   is true when .....
```

Covington is half right when he says that the modes + - ? are "not part of the Prolog language"; the thing is that they WERE part of the DEC-10 Prolog language.
It can be useful to extend the set of modes:

- * (not in Covington) ground on entry
- + nonvar on entry
- - var on entry
- > (not in Covington) thought of as output but might be nonvar
- ? not specified

- = (not in Covington) not necessarily ground, but not further bound

so for example it is simply wrong to write

```
compare(?R: order, +T1: term, +T2: term)
```

because T1 and T2 are allowed to be variables, and

```
compare(?R: order, ?T1: term, ?T2: term)
```

is misleading because we expect that ? arguments will be unified with something, but

```
compare(?R: order, =T1: term, =T2: term)
```

is just right.

In DEC-10 Prolog, - really truly genuinely did mean "MUST be uninstantiated", and code that got that wrong could go seriously insane at run time. Using "-" to mean "normally uninstantiated" is seriously misleading and I wish Covington hadn't recommended that. The difference between > and ? is that a ? argument may be allowed to control what the predicate does, but a > argument should not.

Covington advises that "Comments of this type are not needed for auxiliary predicates" on the grounds that "users of your code need not know about" them. I agree that anything users of your code need to know about should be commented. I do NOT agree that auxiliary predicates do not need to be commented. The convention we used at Quintus was that any auxiliary predicate in a library file which needed commenting got a comment exactly like any other predicate, except that the first line started with "%. " instead of "% ". We had a tool (which I wrote) that extracted comments for a searchable catalogue, and it knew to ignore "%." comments.

Auxiliary predicates in Prolog often correspond to loops in other languages; the meaning of the Prolog predicate is the loop invariant, and really should be stated in a comment.

1.2 Use descriptive argument names in the introductory comment; they need not be the same as those in the clause.
Quibble: they need not, but there should be an extremely obvious family relationship. (Possibly a qualifying prefix or a numeric suffix.) Normally, the names *should* be the same whenever that is practical.

2.1 Make all names pronounceable.
Absolutely!

2.2 Never use two different names that are likely to be pronounced alike.

Also true, although there is much more variation in the pronunciation of English than some Americans appear to realise, so it can be quite hard to tell how things will be pronounced. (I would never have believed that some people would pronounce 'char' as 'care' if I hadn't heard it for myself. I do NOT regard it as reasonable to go out of my way to avoid using both 'char' and 'care'.)

2.3 Construct predicate names with lower-case letters, separating words with underscores.
Absolutely! The single most evil thing about Java was including "_" in the language and then telling people to use baStudlyCaps.

2.4 Do not mix up to, two, and too.
Saving characters was necessary in languages with short identifiers (Fortran up to and including Fortran 77, limit 6; C89 with limit 8 for external identifiers) or file systems with short names (V7 UNIX, 14 characters; MS-DOS, 8+3). Prolog is not such a language.

2.5 Within names, do not express numbers as words.
Quibble: you *can't* make spellings 100% predictable from pronunciations. The route from spellings to pronunciations is about 99% predictable in English, but there are relics of two different scribal traditions and spelling is not 100% predictable from pronunciation. For example, "blue" and "blew" sound the same. I recommend that exported predicates should not have numeric suffixes unless the number is some sort of code number. For example,

```
unicode_4_0_0(?Code, ?Class)
```

might be reasonable.

2.6 Identify auxiliary predicates by appending _aux or _x, _xx, and so forth.
There are no occurrences of that convention in the Quintus library and I am glad of it. _x and _xx are particularly bad; even numbers are better than that. If an auxiliary predicate is there to do a case analysis, '_case' is a better suffix than '_aux'. If an auxiliary predicate is a loop, '_loop' is a better suffix than '_aux'. There is practically *always* something better to use than '_aux'.

I am as lazy as any other programmer. But I did eventually notice that I could *type* a short predicate name and get the computer to expand it to something longer, either by using vi or emacs abbreviations or by simply going back and search/replacing.

The thing that is particularly bad about _x _xx _xxx is that they all look pretty much alike; at least _1 _2 _3 do not resemble each other that much.

2.7 If a predicate represents a property or relation, its name should be a noun, noun phrase, adjective[, adjectival phrase], prepositional phrase (????), or indicative verb phrase.
See Ledgard & Tauer, "Professional Software", for more detailed advice on naming. Do _not_ call everything in sight "is_xxx".

2.8 If a predicate is understood procedurally - that is, its job is to do something rather than to verify a property - its name should be an imperative verb phrase.
Ledgard & Tauer again. The second evil thing about Java is its massive overuse of "get". Do not use "get" for fetching a property.

2.9 Place arguments in the following order: inputs, intermediate results, and final results.
This is a rather oversimplified version of the advice in "The Craft of Prolog".

2.10 Consider how your arguments map onto ordinary English.
Or whatever language you prefer to think in, of course.

3.1 Use descriptive names for variables wherver possible, and make them accurate.
Good naming practice in any language.

3.2 Construct variable names with mixed-case letters, using capitalisation to set off words.
This was the convention that I followed for many years. The longer I live, the less I like it. Covington (and the younger O'Keefe) were right to prefer ResultSoFar to Result_so_far, but I tend to think these days that Result_So_Far is even better.

3.3 For variables of purely local significance, use single letters.
ARGH! Since all variables in Prolog are local to a single clause, surely _all_ variables are "of purely local significance"? I once had the misfortune to have to maintain some code written in this style and I never want to have to do that again.

The question is "how hard is it to read the code". If there is anything tricky about it, give your reader some help.
If you have a file-wide convention that "N is always the total number of elements to process, I is the number of elements processed so far" then you can freely use N and I. If you have a convention that L and U are always the lower and upper bounds of some range, fine. If you have a convention that C0, C1, C2, C are always character codes, S0, S1, S2, S are sequences, and so on, fine. But that convention must *not* be "purely local".

3.4 Use a single letter for the first element of a list and a plural name for the remaining elements.
This is a bent, even broken, version of the usual rule, which is "use a singular name for list elements and the corresponding plural name for the whole list". I regard Covington's [T|Tree] example as *bad* because it is inconsistent. If one thing is a T, why aren't several of them Ts? If several of them are Trees, why isn't one of them a Tree? [Tree|Trees] is best, [T|Ts] is OK (if there is something in the context that helps you figure out that T means Tree), but [T|Tree] is not good at all.

4.1 Use /* */ only to comment out blocks of code; use % for explanatory comments.
I strongly disagree. For one thing, you really should not be 'commenting out', and if you are, /* */ comments don't actually work (as C programmers, if they are any good, are well aware).

In a typical Prolog source file, you will have some sort of standard header with meta-data about the file (name, author, revision, copyright, &c). Then you'll have Prolog :- module declaration and maybe other declarations.

Any then you should have a big comment explaining what the file is all about and what the central ideas are, with maybe a picture of a data structure.

This code should be readable as plain text with a minimum of distracting punctuation. That means that it has to be a /* */ comment, with *no* extra stars. (But possibly dashes at the top and bottom.)
If a file divides naturally into several pieces, each piece can have its own such comment.

4.2 Use layout to make comments more readable.
Use layout to make *anything* more readable.

4.3 If code needs elaborate explanation, consider rewriting it.
Sound advice for any language, but sometimes the result of consideration will be "nope, it's already as simple as we could hope for, it's just a hard problem".

4.4 Indent all but the first line of each clause.
Yes, and begin each clause on a new line.

4.5 If a test is unnecessary because a cut has guaranteed that it is true, say so in a comment at the appropriate place.
I'm not sure that this is always necessary; I am sure that it never hurts.

4.6 Indent an additional 2 spaces between 'repeat' and the corresponding cut.
I agree about indenting, but why 2? I use 4. Also, sometimes the logically final cut is not physically final and shouldn't be.

4.7 Put each subgoal on a separate line, except closely related pairs such as write and nl.
Generally good advice. If you are willing to use format/2, you don't even need to put write and nl on the same line. It's a great pity the ISO Prolog substandard didn't include anything like format/2.

4.8 (A) Skip a line between clauses. (B) Skip two lines between predicates.

- (A) I flatly disagree. Amongst other things, my editor lets me move to the beginning of the previous procedure with one (Meta-Ctrl-R) command (or the next, with Ctrl-_ Meta-Ctrl-R). Sticking in those silly blank lines stuffs this up completely, and it is far too nice a feature to let someone stuff it up.
- (B) I use more blank lines between unrelated predicates than with a group consisting of a predicate and its auxiliaries.

4.9 Keep clauses less than 25 lines if possible.
So they fit on a 24x80 screen, of course. I've never paid much attention to this in languages like Fortran and C. Since in Prolog you *can't* have a loop without splitting out another predicate, and case analyses *often* (but not always) deserve their own predicates too, it is no hardship to follow this rule in Prolog.

4.10 Consider redesigning any non-auxiliary predicate that has more than 4 arguments.
I count roughly 70 such exported predicates in the Quintus library. For example,

```
    %   anti_unify(+Term1, +Term2, -Subst1, -Subst2, -
Term)
    %   binds Term to a most specific generalisation of
Term1 and Term2,
    %   and Subst1 and Subst2 to substitutions such that
    %       Subst1(Term) = Term1
    %       Subst2(Term) = Term2
    %   Substitutions are represented as lists of
Var=Term pairs, where
    %   Var is a Prolog variable, and Term is the term
to substitute for Var.
```

```
    %   When you call it, Subst1, Subst2, and Term
should be variables.
```

This has 2 inputs and 3 outputs; it's hard to see how it could have fewer than 5 arguments, and it only does one thing.

```
    %   ask_number(+Prompt, +Lower, +Upper, +Default,
>Answer)
    %   asks a question where the answer should be a
number in the range
    %   Lower..Upper.  If the user enters an empty
line, the Default
    %   (which need not be in range, or even a number)
is returned.
    %    A typical use might be ask_number
('Percentage', 0, 100, 50, Percentage).
```

This has four inputs and one output. I suppose the four arguments could be packed into a data structure, but that data structure would have no other uses. In fact, it would likely make uses of this predicate *more* complicated, not less, to try to reduce the arity.
I will agree that benchmark:time/15 has far too many arguments, but I didn't write that one.

```
    %   substring(+ABC: text(), ?B: text(),
    %          ?A_Len: integer, ?B_Len: integer, ?
C_Len: integer)
    %   is true when ABC and B are both strings or both
atoms and
    %   there exist texts A and C such that ABC = A ++
B ++ C,
    %   length(A) = Len_A, length(B) = Len_B, length(C)
= Len_C.
    %   The whole string argument (ABC) must be
supplied;
    %   this predicate can solve for all the others.
```

Again, this predicate *needs* 5 arguments.
Also bear in mind the possibility of using term-expansion to supply extra arguments in a systematic way. A non-terminal with 4 arguments is every bit as reasonable as a predicate with 4 arguments, even though a non-terminal with 4

arguments "is" a predicate with 6 arguments (or in some translations, 7 or even 8).

4.11 Consider using a pretty-printer for finished printouts.
The only time a program is "finished" is when its scrapped. If you want fancy printing, use a literal programming tool. I've used fancy listers (this was standard in Interlisp-D), and I came to hate things that changed the font size. What if I wanted to concentrate on the comments rather than the procedure headings? Why would I want the comments in italics or in teeny tiny script or changed so they don't look like comments?
(Having said that, I have a fontiser/colouriser of my own. On very rare occasions it is useful. But for ordinary listsings, it's a pain. It is especially a pain because what you see in the listing DOES NOT MATCH what you see in Emacs.

5.1 Invest appropriate (not excessive) effort in the program; distinguish a prototype ffrom a finished product.
Good advice for any language.

5.2 The most efficient program is the (sic.) one that does the right computation, not the one with the most tricks.
Of course there may be more than one program that does a right computation. The old (Kernighan?) advice "make it simple to make it fast" still holds.

5.3 When efficiency is critical, make tests.
Absolutely!

5.4 Conduct experiments by writing separate small programs, not by mangling the main one.
Nope. Small programs tell you about the performance of small programs, not the real program. The only satisfactory way to tell if a change will help the real program is to change the real program and measure it. Covington seems to be advising people who do not have or do not use any version control system. Make sure the working sources are checked into version control because changing them for "efficiency". Consider making a new version control branch for efficiency experiments.

5.5 Use cuts sparingly but precisely.
Nice advice, but how? See The Craft of Prolog.

5.6 Never add a cut to correct an unknown problem.
Absolutely!

5.7 Avoid the semicolon (;) - make separate clauses instead.

I disagree. His example is very artificial. I originally followed this advice, until it dawned on my that most of the time there *wasn't* any natural concept and it was extremely hard to give the case analysis predicate any sensible name.
The important thing here is not avoiding semicolons but giving things their own identities and names whenever there is a helpful name that they *can* be given.

5.8 Use parentheses whenever operator precedence is important: do not assume that people have memorised the precedence table.
Hmm. Why don't we just abandon operator syntax entirely then and just use Lispy syntax? (Actually, I *like* Lispy syntax.)
In the specific case of x, y ; z I agree that parentheses are advisable, but I'd write it as

```
(    x,
      y
      ;    z
      )
```

where the layout provides the important clue about precedence I find code using otiose parentheses like (x, y) ; z disgusting and difficult to read.
The important thing here is

- whenever there are so many or such rare operators in a part of the program that people might plausibly be confused, rewrite it.

5.9 When you use ; always use parentheses
But around the *whole* form, *not* around the conjunctions.

5.10 Avoid if-then-else structures.u
I strongly disagree. Very strongly indeed. If-then-elses are MUCH better than cuts. This is especially true in Mercury, of course.

5.11 Look out for constructs that are almost always wrong.
True.

5.12 Work at the beginning of the list
True, but don't be fanatical about it.

5.13 Avoid append as far as possible.
Use append liberally in *design*; use difference pairs or grammar rules in *implementation*.

5.14 Use difference lists to achieve fast concatenation.
Quibble: I regard the term "differnce lists" as misleading. I use "difference pairs" (of arguments) or "list differences". Apart from that, fine.
There are other ways to get fast concatenation, of course.

5.15 Use tail recursion for recursions of unknown depth.
I don't see what "unknown depth" has to do with it. I would say "prefer tail recursion to body recursion, but don't be scared of body recursion if you need it".

5.16 Recognise that tail recursion is unimportant when the depth of the recursion is limited to about 50 or less.
I agree that you should not "sacrifice simplicity and clarity" to achieve microscopic gains, but I have never found it useful to ask "is this recursion limited to about 50 or less". Even for short recursions (and what counts as "short" on a 1GB machine is different from what used to count as "short" on 1MB machiens) tail recursion is often the most natural way to express things.
I'd say that the rule is "use tail recursion whenever it is a clear way to say what you want, WHATEVER the likely depth of recursion; use body recursion if that's the easiest way to get the code right, WHATEVER the likely depth of recursion". Worry about restructuring *after* you have measured what the code is actually doing.

5.17 Avoid assert and retract unless you actually need to preserve information through backtracking.
I would also say "If you have a dynamic predicate, write interface predicates for changing it instead of using 'bare' calls to assert and retract, so that your interface predicates can check that the data base will still be logically consistent after the change."

5.18 For sorting, use merge sort or a built-in sorting algorithm.
He has the details wrong: Quicksort is *NOT* O(n log n), it is O(n^2) and this worst case _often_ happens in practice. (Yes, I know about the arguments in textbooks that say it's extremely unlikely for uniform random inputs; the problem is that real-world inputs are not uniform random inputs.)

5.19 Instead of sorted lists, consider using trees.
"A binary tree never needs sorting because it is never out of order"; however, it DOES need work, sometimes rather complicated work, to KEEP it in order. In a WAM-based implementation where a list requires 2 words per element and other compound terms with N arguments require N+1 words, a tree will cost twice as much storage as a list.

The *right* rule is "use binary trees if you need element-wise access; use sorted lists if you can do most things on whole collections."

6.1 Isolate non-portable code.
Good advice for any language.

6.2 Isolate "magic numbers".
In fact the example of pi is a very good one. I once had to use a Prolog system from one of our competitors where one part of their library used a 64-bit value for pi and another part used an 80-bit value for pi. Not surprisingly, trig code got rather messed up.

6.3 take the extra minute to prevent errors rather than having to find them later.
Good advice for any language.

6.4 Test code at its boundaries (limits).
Good advice for any language.

6.5 Test that each loop starts correctly, advances correctly, and ends correctly.
Good advice for any language; could go further.

6.6 Test every predicate by forcing it to backtrack.
Misleadingly worded: many predicates *can't* backtrack. "Test every predicate by failing back into it"; now that's something you *can* do.

6.7 Test predicates by supplying arguments of the wrong types.
I'm not so sure about this one. For a beginner, it may be useful advice so that they find out what is likely to provoke which error message. But append/3, for example, is designed to work with lists; it isn't *intended* to do anything in particular if given arguments of any other type, so there is no sense in which you can test it for other types. (To test something, there must be a specified set of acceptable behaviours so that you can tell whether the actual behaviour is acceptable or not. If "anything goes", then the "test" cannot fail.)

6.8 Do not waste time testing for errors that will be caught anyhow.
That'd be nice, except that we do not in general *know* which errors "will be caught anyhow". 6.8 appears to contradict 6.7.

6.9 In any error situation, make the program either correct the problem or crash (not just fial).
True. "Crashing" here means calling raise_exception/1 (which the ISO Prolog substandard very confusingly renamed to throw/1).

6.10 Make error messages informative.
Good advice, but error reports should be related to the exported predicate they originated from, *not* to some internal predicate which the user doesn't know about.
When we added exception handling to Quintus Prolog, some built-in predicates were given error handlers just so that they could handle internal deeply generated exceptions and map them back to user-level descriptions.

6.11 Master the Prolog debugger; it is simple, powerful, and portable.
Simple, yes. Powerful, yes, although it's even better with the advice package. Portable, not as much as it could be. There are 3-port, 4-port, 5-port, and even 7-port debuggers out there, and that's without considering exceptions. I find it difficult to use the SWI debugger because the keystrokes that years of using Quintus Prolog programmed into my fingers don't work in SWI. (One of the reasons I'd rather --disable-readline.)

6.12 Use write for debugging.
Wrong. Use print for debugging; it was specifically intended for that.

6.13 Mark all temporarily altered lines of code with "%TEST!!!".
I don't find this very useful; if I want to know what I changed, I just ask SCCS.

6.14 Use write(!!!) to mark places in the program where work remains to be done.
I've never found that particularly useful either.
Something slightly more useful is

```
shouldnt :-
    write(user, 'This should not happen'), nl(user),
    break,
    throw(shouldnt).
```

(adapted from Interlisp).

From: **Michael Maxwell**
Subject: Re: Good Examples of Properly Commented Prolog Code

Richard O'Keefe wrote:

```
> 6.13 Mark all temporarily altered lines of code with "%
TEST!!!".
>
>     I don't find this very useful; if I want to know what
```

```
I changed,
>     I just ask SCCS.
>
> 6.14 Use write(!!!) to mark places in the program where
work remains
>     to be done.
>
>     I've never found that particularly useful either.
```

Sometimes these comments persist through multiple revisions of a file, so SCCS or RCS aren't useful ways to point them out.
FWIW, my favorite editor (Visual SlickEdit) allows color coding of constructs on a programming language-specific basis. Most editors these days do (including jEdit).

What dawned on me awhile back is that my editor allows me to define user-specific tokens, with their own color coding, and I can use that to put tokens like '% TEST!!!' or '%!!!' in some color that stands out, like red. That way temporarily altered lines, or things that I want to fix but don't have the time to do right now, are easily spotted when I scroll through a buffer. (If I used TreeWare, and a color printer, this would work too.)

This color coding of course doesn't work for someone else who is reading my file, unless we standardize the user-defined tokens. But Michael Covington's suggested use of '!!!' at least provides some visual cue.

From: **Jan Wielemaker**
Subject: Re: Good Examples of Properly Commented Prolog Code

Richard A. O'Keefe wrote:

```
> Steve Moyle wrote:
>> One place to start would be Covington's guidlines:
>>
>> [PDF] Some Coding Guidelines for Prolog
>> http://www.ai.uga.edu/mc/plcoding.pdf
>
> I have a few quibbles with that.
```

Not sure my understanding of `a few' is the same as yours :-)

```
>     The form I use begins with
>
>     %   <name>(<mode><var>: <type>, ..., <mode><var>:
<type>)
>     %   is true when .....
```

I like this very much, except I'd like to have a blank line between the first semi-formal line and the description, starting with % Sofar I didn't use types, but I might start doing that. For a long time I plan to add some code to the help-system that makes this documentation available in a transparent way. Still didn't find the time to do it :-(Here are is a typical example of what I use

```
    %    age(?Name, ?Age)
    %
    %    True if person named Name has age Age.
```

```
> 2.6 Identify auxiliary predicates by appending _aux or _x,
_xx,
>     and so forth.
>
>     There are no occurrences of that convention in the
Quintus library
>     and I am glad of it.  _x and _xx are particularly bad;
even numbers
>     are better than that.  If an auxiliary predicate is
there to do
>     a case analysis, '_case' is a better suffix than
'_aux'.  If an
>     auxiliary predicate is a loop, '_loop' is a better
suffix than
>     '_aux'.  There is practically *always& something
better to use
>     than '_aux'.
>
>     I am as lazy as any other programmer.  But I did
eventually notice
>     that I could *type* a short predicate name and get the
computer to
>     expand it to something longer, either by using vi or
emacs
>     abbreviations or by simply going back and search/
```

```
replacing.
>
>     The thing that is particularly bad about _x _xx _xxx
is that they
>     all look pretty much alike; at least _1 _2 _3 do not
resemble each
>     other that much.
```

Not so sure. I still struggle with the naming of auxiliary predicates. _loop is nice if it applies, but quite often there is no sensible name to give to them. _x* works fine to about 3-4 levels, which is often enough. To me _xx and _2 are about equal.

```
>     This was the convention that I followed for many
years.  The longer
>     I live, the less I like it.  Covington (and the
younger O'Keefe)
>     were right to prefer ResultSoFar to Result_so_far, but
I tend to think
>     these days that Result_So_Far is even better.
```

I prefer ResultSoFar.

```
> 4.6 Indent an additional 2 spaces between 'repeat' and the
corresponding
> cut.
>
>     I agree about indenting, but why 2?  I use 4.
>     Also, sometimes the logically final cut is not
physically final
>     and shouldn't be.
```

I used to do that, but since the days of auto-indenting editors things get more tricky. Because where I like this for repeat, it should apply to any generator in a failure-driven loop. I used to write

```
        (   member(X, List),
            process(X),
            fail
        ;   true
        )
```

But even PceEmacs cannot determine when this layout is appropriate and I hate doing the indentation by hand. So I stopped using this. Today I use something like this:

```
        (   repeat,
            action,
            condition
        ->  true
        ).
```

Where applicable, use forall(generator(X), action(X)). It also avoids failure of action/1 go unnoticed. Better a program saying 'No' than producing the wrong result.

```
> 4.7 Put each subgoal on a separate line, except closely
related pairs
>     such as write and nl.
>
>     Generally good advice.  If you are willing to use
format/2, you
```

True, but quite a few interesting systems have it and the implementations are reasonable compatible.

```
> 4.8 (A) Skip a line between clauses.
>     (B) Skip two lines between predicates.
>
>     (A) I flatly disagree.  Amongst other things, my
editor lets me move
>         to the beginning of the previous procedure with
one (Meta-Ctrl-R)
>         command (or the next, with Ctrl-_ Meta-Ctrl-R).
Sticking in
>         those silly blank lines stuffs this up completely,
and it is far
>         too nice a feature to let someone stuff it up.
```

I agree with Richard. Next clause on next line, next related predicate one blank line and next unrelated predicate two blank lines works just fine. The not indented clause head is enough to seperate the clauses.

```
> 4.9 Keep clauses less than 25 lines if possible.
>
>     So they fit on a 24x80 screen, of course.
>
>     I've never paid much attention to this in languages
like Fortran and
>     C.  Since in Prolog you *can't* have a loop without
splitting out
>     another predicate, and case analyses *often* (but not
always)
>     deserve their own predicates too, it is no hardship to
follow this
>     rule in Prolog.
```

Unless you have I/O like (including graphics calls) predicates ... I've seen _very_ long clauses, even overflowing the 512 subclause limit the system had very long ago (now unlimited).

```
> 4.11 Consider using a pretty-printer for finished
printouts.
>
>     The only time a program is "finished" is when its
scrapped.
>     If you want fancy printing, use a literal programming
tool.
>     I've used fancy listers (this was standard in
Interlisp-D),
>     and I came to hate things that changed the font size.
What
>     if I wanted to concentrate on the comments rather than
the
>     procedure headings?  Why would I want the comments in
italics
>     or in teeny tiny script or changed so they don't look
like
>     comments?
>
>     (Having said that, I have a fontiser/colouriser of my
own.
>     On very rare occasions it is useful.  But for ordinary
listsings,
```

```
>     it's a pain.  It is especially a pain because what you
see in
>     the listing DOES NOT MATCH what you see in Emacs.
```

Might be a bit outdated. Who prints programs these days with nice large colour displays? I've not printed code for years. Do get yourself a 1280x1024 or bigger monitor. Much better investment than a faster CPU.

```
> 5.3 When efficiency is critical, make tests.
>
>     Absolutely!
```

Use the profiler. Use the graphical debugger to examine choice-points and see whether they are as you expect them to be. Be aware that tests are good, but performance depends on the Prolog environment. I try to ensure performance of a particular construct does not deteriorate much in later versions, but sometimes an alternative construct may become much faster.

```
> 5.5 Use cuts sparingly but precisely.
>
>     Nice advice, but how?  See The Craft of Prolog.
```

True.

```
> 5.10 Avoid if-then-else structures.
>
>     I strongly disagree.  Very strongly indeed.  If-then-
elses are
>     MUCH better than cuts.  This is especially true in
Mercury, of course.
```

I tend to use a predicate if I can give it a meaningful name, i.e. if by giving it a name the readability of the main clause improves. Otherwise use if-then-else.

```
> 5.17 Avoid assert and retract unless you actually need to
preserve
>     information through backtracking.
>
>     I would also say "If you have a dynamic predicate,
write interface
>     predicates for changing it instead of using 'bare'
```

```
calls to
>     assert and retract, so that your interface predicates
can check
>     that the data base will still be logically consistent
after the
>     change."
```

These seems more complimentary than conflicting statements. Both are true. Using assert/retract as a substitute for global variables is a really bad idea. It often harms performance, it make the nice `retry' option of the debugger worthless, it has trouble with cleanup, re-entrance and threads. If you really need them, write an interface as Richard suggests. Especially true in multi-threaded applications as you probably need explicit synchronisation using mutexes.

```
> 6.4 Test code at its boundaries (limits).
>
>     Good advice for any language.
```

Time for a good documented testing skeleton ... There are various out there of different quality.

```
> 6.11 Master the Prolog debugger; it is simple, powerful,
and portable.
>
>     Simple, yes.  Powerful, yes, although it's even better
with the
>     advice package.  Portable, not as much as it could
be.  There are
>     3-port, 4-port, 5-port, and even 7-port debuggers out
there, and
>     that's without considering exceptions.  I find it
difficult to use
>     the SWI debugger because the keystrokes that years of
using Quintus
>     Prolog programmed into my fingers don't work in SWI.
(One of the
>     reasons I'd rather --disable-readline.)
```

It took me very long, mostly because after I write it hardware was too slow, but these days I rarely touch the normal debugger anymore. 99% of the cases I use the graphical one. Unfortunately sometimes it goes nuts :-(

```
> 6.12 Use write for debugging.
>
>     Wrong.  Use print for debugging; it was specifically
intended for that.
```

use debug(Channal, Format, Arguments). You can leave them in your code with no overhead if you compile using -O and they act as comments too.

```
> 6.13 Mark all temporarily altered lines of code with "%
TEST!!!".
>
>     I don't find this very useful; if I want to know what
I changed,
>     I just ask SCCS.
```

True. I strongly dislike files with commented dead code. Only, use CVS or subversion :-)

```
> 6.14 Use write(!!!) to mark places in the program where
work remains
>     to be done.
>
>     I've never found that particularly useful either.
>
>     Something slightly more useful is
>
>     shouldnt :-
>         write(user, 'This should not happen'), nl(user),
>         break,
>         throw(shouldnt).
>
>     (adapted from Interlisp).
```

Most of the time I put % TBD: whatever at the end of the line. Sometimes I simply call tbd('Whatever'). That will cause the system to raise an undefined predicate. Even better, list_undefined/0 will report them and where they are called.

From: **Richard O'Keefe**
Subject: Re: Good Examples of Properly Commented Prolog Code

```
Jan Wielemaker wrote:
> I like this very much, except I'd like to have a blank
line between the
> first semi-formal line and the description, starting with %
```

When the comment starts out like this:

```
          %   <name>(<mode><var>: <type>, ..., <mode><var>:
     <type>)
          %   is true when .....
```

the first line is the subject of the first sentence. It is really very jarring to put a blank line in the *middle* of a sentence.

```
> %    age(?Name, ?Age)
> %
> %    True if person named Name has age Age.
```

I would write that as

```
    %   age(?Name: atom, ?Age: number)
    %   is true when Name is the name of a person
    %   and the age of that person in years is Age.
```

or without the types as

```
    %   age(?Name, ?Age)
    %   is true when Name is the name of a person (as an
atom)
    %   and the age of tha person in years is Age (as a
number).
```

In this case, I might leave out the (as a number) bit; what else would it be? But the encoding of the name is _not_ obvious and needs saying. A general convention within a file that names are atoms (as opposed to character lists or strings or even name(Family,Personal) or whever) would mean you don't have to say it every time.

```
>> 2.6 Identify auxiliary predicates by appending _aux or
_x, _xx,
>>     and so forth.
>
```

```
> Not so sure.  I still struggle with the naming of auxiliary
> predicates.
```

Why, so do I.

```
> _loop is nice if it applies, but quite often there is no
> sensible name to give to them.  _x* works fine to about 3-4
> levels, which is often enough.  To me _xx and _2 are about
> equal.
```

I disagree. I don't find that _x works to *any* number of levels, and I have real trouble with the idea of 4 levels of "miscellaneous" auxiliary predicates in the first place. Given that one of the first Prolog programs I worked on was a geometry theorem prover (I didn't write the thing, I just revised it), I still think of _x as referring geometric co-ordinate and expect the next two in the series to be _y and _z.

Perhaps we could advance this discussion further if Jan provided an example where he wanted to use _x _xx and _xxx and see what I can do about naming them.

```
>>     This was the convention that I followed for many
years.  The longer
>>     I live, the less I like it.  Covington (and the
younger O'Keefe)
>>     were right to prefer ResultSoFar to Result_so_far,
but I tend to think
>>     these days that Result_So_Far is even better.
>
> I prefer ResultSoFar.
```

As I said, I _used_ to. These days, I am less tolerant of things that violate the normal spacing rules for text.

```
>> 4.6 Indent an additional 2 spaces between 'repeat' and
the corresponding
>> cut.
>>
>>     I agree about indenting, but why 2?  I use 4.
>>     Also, sometimes the logically final cut is not
physically final
```

```
>>     and shouldn't be.
>
> I used to do that, but since the days of auto-indenting
editors
```

When I first heard about Emacs, I was ever so keen. When I actually got my hands on Emacs, it was a real disappointment. Emacs' auto-indentation for C was *horrible*, and it turned out to be impossible to reach a tolerable style by fiddling with its parameters. To this day, one of the first things I do with an autoindenting editor is switch automatic indentation *OFF*.

The editor I use is Emacs-*like*. For indentation,

Return -> make a new line with no indentation below the
current line, move the cursor to the new line
^_ Return -> like Return, but new line is above old line,
Line feed -> make a new line with the same indentation as the
current line, below the current line, and move
the cursor to it (just after the indentation)
^_ Line feed-> like Line feed, but new line is above old line.
Meta ^I -> indent current line by one step (default 4,
settable by ^U n Meta +)
^U n $^I -> indent by n steps
Meta ^U -> outdent current line by one step
^U n $^U -> outdent by n steps
Meta i -> indent current region (mark to point) one step
^U n $i -> indent current region n steps
Meta u -> outdent current region (mark to point) one step
^U n $u -> outdent current region n steps

```
> things get more tricky.
```

Indeed things get tricky. That's because all the autoindenters I've come across are far more trouble than they are worth.

```
> Because where I like this for repeat, it should apply
> to any generator in a failure-driven loop.  I used to write
>
>     (   member(X, List),
>            process(X),
>         fail
```

```
>      ;   true
>      )
```

OK, how would we do that in my editor?

```
   "(   member(X, List)," ^J          21
   $^I $^I "process(X)," ^J          16
   $^U "fail" ^J                  7
   $^U ";   true" ^J             11
   ")"                    1
              Total:       56 keystrokes.
```

Who needs an auto-indenter? Hmm. I've just added a very simple auto-indent feature. If enabled,

"(" at the start of a line adds indent-1 spaces after it;
";" at the start of a line indents, adds ";", and indent-1 spaces;
"->" at the start of a line indents, adds "->", and indent-2 spaces;
")" at the start of a line outdents unless the previous line began with "(" ";" or "->".
LF indents the new line one step if the previous line starts with "(" ";" or "->".

So we get

```
   "(member(X, List)," ^J             18
   $^I "process(X)," ^J             14
   $^U "fail" ^J                  7
   ";true" ^J                  6
   ")"                    1
              Total:       46 keystrokes.
```

Autoindenting doesn't save all that many keystrokes.

But I actually disagree. I do NOT think "it should apply to any generator in a failure-driven loop". There is something very very special about repeat/0 which does NOT apply to member/2 (at least in normal use), namely that repeat/0 never fails. It has and is intended to have infinitely many solutions. My layout for failure-driven loops is

```
    (   member(X, List)
        process(X),
        fail ; true
```

```
    ),
```

just try and teach _that_ to an auto-indenter. Note the signature of a failure-driven loop: "fail ; true". A repeat loop does NOT have that signature. A repeat loop has to be terminated by some kind of cut.

```
> Where applicable, use forall(generator(X), action(X)).
```

Agreed.

```
> Better a program saying 'No' than producing the wrong
result.
```

Absolutely!

```
>> 4.9 Keep clauses less than 25 lines if possible.
>>
>>     So they fit on a 24x80 screen, of course.
>>
>>     I've never paid much attention to this in languages
like Fortran and
>>     C.  Since in Prolog you *can't* have a loop without
splitting out
>>     another predicate, and case analyses *often* (but not
always)
>>     deserve their own predicates too, it is no hardship
to follow this
>>     rule in Prolog.
>
> Unless you have I/O like (including graphics calls)
predicates
> ... I've seen _very_ long clauses, even overflowing the 512
> subclause limit the system had very long ago (now
unlimited).
```

At Quintus we had a bug report from a customer: they had a clause with more than 64000 variables! It was machine-generated, of course.

```
>> 4.11 Consider using a pretty-printer for finished
printouts.
>
> Might be a bit outdated.  Who prints programs these days
```

```
with
> nice large colour displays?
```

Me.

```
> Do get yourself a 1280x1024 or bigger monitor.
```

I *have* a 1280x1024 monitor. In fact both my SunBlade100 and my G4 Mac have such monitors. You *still* don't get a whole lot of text on them, and if you want to really carefully read a file, paper is *wonderful*. I tend to find that syntax colouring just makes stuff less readable (because it reduces the contrast, which is well known to reduce readability).

```
>> 5.10 Avoid if-then-else structures.
>>
>>     I strongly disagree.  Very strongly indeed.  If-then-
elses are
>>     MUCH better than cuts.  This is especially true in
Mercury, of course.
>
> I tend to use a predicate if I can give it a meaningful
name, i.e. if by
> giving it a name the readability of the main clause
improves. Otherwise
> use if-then-else.
```

A difference of emphasis here, but we actually agree. Finding it hard to assign a name to something is a good clue that you should not be splitting it out.

```
>> 5.17 Avoid assert and retract unless you actually need to
preserve
>>     information through backtracking.
>>
>>     I would also say "If you have a dynamic predicate,
write interface
>>     predicates for changing it instead of using 'bare'
calls to
>>     assert and retract, so that your interface predicates
can check
>>     that the data base will still be logically consistent
after the
```

```
>>     change."
>
> These seems more complimentary than conflicting statements.
```

They are complementary, that's why I said "I would ALSO say"...

```
>> 6.12 Use write for debugging.
>>
>>     Wrong.  Use print for debugging; it was specifically
intended for that.
>
> use debug(Channal, Format, Arguments). You can leave them
in your code
> with no overhead if you compile using -O and they act as
comments too.
```

Good advice for SWI Prolog. The point I was making is that print/1 exists so that output can be tailored (and in particular, heavily and appropriately abbreviated) for debugging, while write/1 insists on showing you everything. Combine the advice: if you use debug/3, use the ~p format rather than the ~w format.

```
> True.  I strongly dislike files with commented dead code.
Only, use
> CVS or subversion :-)
```

For a single person project, SCCS or RCS are pretty much ideal. The CVS documentation has me completely baffled.

```
> Most of the time I put % TBD: whatever at the end of the
line. Sometimes
> I simply call tbd('Whatever'). That will cause the system
to raise an
> undefined predicate. Even better, list_undefined/0 will
report them and
> where they are called.
```

The problem with comments like % TBD is that they are still there 4 years and 7 releases later. tbd/1 and list_undefined/0 are excellent (albeit SWI-specific) advice.

From: **Jan Wielemaker**
Subject: Re: Good Examples of Properly Commented Prolog Code

We're getting close to consensus :-)

Richard A. O'Keefe wrote:

```
> Jan replied to my comments on Covington.
>     I like this very much, except I'd like to have a blank
line between the
>     first semi-formal line and the description, starting
with %
>
> When the comment starts out like this:
>      %   <name>(<mode><var>: <type>, ..., <mode><var>:
<type>)
>      %   is true when .....
> the first line is the subject of the first sentence.
> It is really very jarring
```

I see. I prefer to see it as a synopsis like the traditional Unix manpages. One of the reasons is different patterns. What about

```
    %    age(+Name, -Age)
    %    age(-Name, +Age)
    %
    %    ....
```

Are you going for

```
    %    age(+Name, -Age)
    %    is true if ...
    %
    %    age(-Name, +Age)
    %    is true if ...
```

I also tend to document different arities in the same comment, like this:

```
    %    rdf_load(+File)
    %    rdf_load(+File, +Options)
    %
    %    ....
```

```
    rdf_load(File) :-
        rdf_load(File, []).

    rdf_load(File, Options) :-
        ....
```

```
> Perhaps we could advance this discussion further if Jan
provided an
> example where he wanted to use _x _xx and _xxx and see
what I can do
> about naming them.
```

I often call the do_something, or whatever. The issue frequently arrises on constructs as below. In this particular case load_stream/1 would be a good name, but there are enough cases where it is much harder to find a good name that isn't the same as the name of the main predicate.

```
    load_file(File) :-
        open(File, read, In),
        call_cleanup(do_load_file(In), close(In)).

    do_load_file(...)
```

```
>     I used to do that, but since the days of auto-
indenting editors
>
> When I first heard about Emacs, I was ever so keen.
> When I actually got my hands on Emacs, it was a real
disappointment.
> Emacs' auto-indentation for C was *horrible*, and it
turned out to be
> impossible to reach a tolerable style by fiddling with its
parameters.
> To this day, one of the first things I do with an
autoindenting editor
> is switch automatic indentation *OFF*.
```

Properly trained, which unfortunately generally means sticking to the style used by the author of the indentation module or writing your own, auto-indenting editors are very valuable, as the fact that the indentation goes wrong is an immediate clue you've made a mistake. For me its 'can't program without them'. Again, its not

about the number of keystrokes (although it helps a little), its about the feedback on errors.

```
> But I actually disagree.  I do NOT think "it should apply
to any
> generator in a failure-driven loop".  There is something
very very
> special about repeat/0 which does NOT apply to member/2
(at least
> in normal use), namely that repeat/0 never fails.  It has
and is
> intended to have infinitely many solutions.  My layout for
```

Partly true, but in C I use the same layout for "for(;;)" and "for(init;cond;next)". I don't see why this isn't the case in Prolog. They are equally special and dangerous.

```
> failure-driven loops is
>
>     (   member(X, List)
>         process(X),
>         fail ; true
>     ),
>
> just try and teach _that_ to an auto-indenter.  Note the
signature
```

Works perfect in PceEmacs. I used to use fail ; true this way. Don't really know why I stopped doing it, maybe I should re-introduce it.

```
>     Unless you have I/O like (including graphics calls)
predicates
>     ... I've seen _very_ long clauses, even overflowing
the 512
>     subclause limit the system had very long ago (now
unlimited).
>
> At Quintus we had a bug report from a customer:  they had
a clause
> with more than 64000 variables!  It was machine-generated,
of course.
```

Thats normal. Almost all limits the initial compiler had (#subclauses, #variables, size of produced code) had to be removed over time. The current compiler is limited by memory only, as most of the rest of the system.

```
>     > 4.11 Consider using a pretty-printer for finished
printouts.
>
>     Might be a bit outdated.  Who prints programs these
days with
>     nice large colour displays?
>
> Me.
>
>     Do get yourself a 1280x1024 or bigger monitor.
>
> I *have* a 1280x1024 monitor.  In fact both my SunBlade100
and my G4 Mac
> have such monitors.  You *still* don't get a whole lot of
text on them,
> and if you want to really carefully read a file, paper is
*wonderful*.
> I tend to find that syntax colouring just makes stuff less
readable
> (because it reduces the contrast, which is well known to
reduce
> readability).
```

I must say I find it hard these days to read Prolog without the colouring of PceEmacs. Guess its what you're used to. Just keyword colouring isn't very valuable in Prolog.

```
>>> 6.12 Use write for debugging.
>>>
>>>     Wrong.  Use print for debugging; it was specifically
intended for
>>> that.
>>
>>use debug(Channal, Format, Arguments). You can leave them
in your code
>>with no overhead if you compile using -O and they act as
```

```
comments too.
>
> Good advice for SWI Prolog.  The point I was making is
that print/1
```

Its easy enough to port the library to any system providing term_expansion or -bit less work- goal_expansion.

```
>> True.  I strongly dislike files with commented dead
code.  Only, use
>> CVS or subversion :-)
>
> For a single person project, SCCS or RCS are pretty much
ideal.
> The CVS documentation has me completely baffled.
```

You need very little for single person development. cvs add file, cvs rm file, cvs commit, cvs update and cvs diff cover 99% of what you need. Install web-browsing of the repository and you're all set and done.

```
>> Most of the time I put % TBD: whatever at the end of the
line. Sometimes
>> I simply call tbd('Whatever'). That will cause the system
to raise an
>> undefined predicate. Even better, list_undefined/0 will
report them and
>> where they are called.
>
> The problem with comments like % TBD is that they are
still there
> 4 years and 7 releases later.  tbd/1 and list_undefined/0
are excellent
```

True, but if the problem never became serious who cares?

```
> (albeit SWI-specific) advice.
```

Or any other system with a cross-referencer.

From: **Uwe Lesta**
Subject: Re: Good Examples of Properly Commented Prolog Code

I follow the discussion with a half eye.

Yes, Coding Guidelines are useful.

They are more useful if other tools like pretty_printer work with it.

but, IMO they will be commonly used if it is simple to write them and i will get immediately benefit of it.

A good sample for comments of methods are from ms visual studio The CSharp editor has templates for comments which are automatically inserted after typing '///' in the line befor a method definition.

and you get 'tooltips' shown by this comments if you point with the mouse on a method as well as you type the parameter list.

If you haven't seen it, try to get a lock on it.

My other point about no implicit conversion is this: If rationals stay rational, I don't think we need to worry about sort et. al being extended to support them. Use predsort combined with `compare_rationals/3`, because mixed data will not be expected.

From: **Richard O'Keefe**
Subject: Re: Good Examples of Properly Commented Prolog Code

Jan Wielemaker wrote:

```
> I see.  I prefer to see it as a synopsis like the
traditional Unix manpages.
```

Manpages are good for Manual PAGES. That kind of layout is not designed for and not particularly good for SHORT comments.

```
> One of the reasons is  different patterns.  What about
>
> %    age(+Name, -Age)
> %    age(-Name, +Age)
```

```
> %
> %    ....
>
> Are you going for
>
> %    age(+Name, -Age)
> %    is true if ...
> %
> %    age(-Name, +Age)
> %    is true if ...
```

No. ALL patterns should be TRUE in the same cases, otherwise why do they have the same name? In that case, I'd leave the modes out of the first line and write

```
     %   age(Name, Age)
     %   is true when ....
     %   This may be used in modes (+,-) and (-,+) but no
others.
```

```
> I also tend to document different arities in the same
comment,
```

I don't. They are different predicates. They may be related, but they are different.

Note that we are talking here about the leading comment for ONE PREDICATE. If I want to say something about a GROUP of predicates, I have ANOTHER comment ahead of the group which describes the group.

```
>> But I actually disagree.  I do NOT think "it should apply
to any
>> generator in a failure-driven loop".  There is something
very very
>> special about repeat/0 which does NOT apply to member/2
(at least
>> in normal use), namely that repeat/0 never fails.  It has
and is
>> intended to have infinitely many solutions.  My layout for
>
> Partly true, but in C I use the same layout for "for(;;)"
and
```

```
> "for(init;cond;next)".
```

The less I say about your C layout, the better. If I said what I really think about that, you would probably bar me from the list for life.The thing is that in C it's the *same* construct:

```
    for (init; test; step) {
        body1;
        if (test2) break;
        body2;
        ...
    }
```

```
> I don't see why this isn't the case in Prolog.
> They are equally special and dangerous.
```

No, I have already explained why they are NOT equally dangerous.

```
> I must say I find it hard these days to read Prolog
without the colouring
> of PceEmacs.  Guess its what you're used to.  Just keyword
colouring isn't
> very valuable in Prolog.
```

I have seen, I have tried, keyword colouring in Prolog, and find that it distracts me from the content I need to see and makes the code nearly unreadable. Prolog has so few keywords, and there is so little "key" about them, that there seems to me to be practically nothing gain by highlighting them.

From: **Michael Maxwell**
Subject: Re: Good Examples of Properly Commented Prolog Code

Richard A. O'Keefe wrote:

```
> I have seen, I have tried, keyword colouring in Prolog,=20
> and find that it distracts me from the content I need=20
> to see and makes the code nearly unreadable.  Prolog=20
> has so few keywords, and there is so little "key" about=20
> them, that there seems to me to be practically nothing=20
> gain by highlighting them.
```

The times I've been saved by my editor's coloring are times when I forgot to terminate a comment or a string. I don't recall whether this was in Prolog or some other language, but it definitely helped find one class of errors.

From: **Richard O'Keefe**
Subject: Re: Good Examples of Properly Commented Prolog Code

Michael Maxwell wrote:
```
> The times I've been saved by my editor's coloring are
times when I =
> forgot to terminate a comment or a string.
```

The funny thing is that nobody ever seems to cite any other ways that it helps. There are other and arguably better tools for the job. For example, my editor includes a "Check Prolog syntax" command (Meta-_) which checks the syntax of the next clause. It picks up unterminated comments by noticing /* inside a comment or by having too long a comment; it can pick up unterminated strings and atoms by noticing newlines inside strings that are not part of a continuation sequence. It can *also* find a heck of a lot of other things that colouring will never notice.

The big thing about syntax colouring is that it tends to restrict you to having ONE syntax in a file. If you like literate programming or have examples of one language inside another, you suffer.

From: **Richard O'Keefe**
Subject: Re: Good Examples of Properly Commented Prolog Code

I wrote:
```
> Perhaps we could advance this discussion further if Jan
provided an
> example where he wanted to use _x _xx and _xxx and see
what I can do
> about naming them.
```

Jan Wielemaker replied:

```
> I often call the do_something, or whatever.  The issue
> frequently arrises on constructs as below.  In this
particular
> case load_stream/1 would be a good name, but there are
enough
> cases where it is much harder to find a good name that
isn't the
> same as the name of the main predicate.
>
> load_file(File) :-
>     open(File, read, In),
>     call_cleanup(do_load_file(In), close(In)).
>
> do_load_file(...)
```

I've been using Lisp long enough to find the name 'unwind_protect' more immediately comprehensible than 'call_cleanup', but never mind. Actually, no. We SHOULD mind. Because if we use the name 'unwind_protect' there is an immediately obvious name for the subordinate predicate:

```
    %   load_file(*File: file_name)
    %   is given a file name and is responsible for loading
it.
    %   Whatever happens, the stream should not be left open.

    load_file(File) :-
        open(File, read, In),
        unwind_protect(protected_load_stream(In), close(In)).

    %.  protected_load_stream(*In: input_stream)
    %   is given a stream open for reading; it loads forms
    %   from that stream.  It is the responsibility of the
    %   caller to close the stream; this predicate may raise
    %   exceptions to report problems.

    protected_load_stream(In) :-
        ...
```

Here there are two changes to the name:

1. load_<<file>> becomes load_<<stream>>, because the one is given a file name, and the other is given a stream (which need not ever have had anything to do with a file, so ...load_file... would be a bad name for it.
2. <<>>load_stream becomes <<protected_>>load_stream to make it clear that this should only be called in the scope of an unwind_protect/2 that will do whatever cleanup is necessary. If you insist on call_cleanup/2 (which I think is a bad name because it is not the *call* that needs a cleanup action but the *exit, failure, or exception*) then you could turn load_stream<<>> into load_stream<<_no_cleanup>>.

```
> Properly trained, which unfortunately generally means
sticking to the
> style used by the author of the indentation module
```

Yes, but authors of indentation modules seem to come up with such execrably bad indentation styles.

```
> or writing your own,
```

I've tried that too. The thing that I found when I did it is that RIGID INDENTATION STYLES ARE WRONG. For example, it often pays to lay things out in a tabular style. Auto-indenters destroy that. In fact, that's one of the standing complaints about indent(1).

To give a trivial example,

```
    struct {
        char *name;
        int   day;
        int   month;
    } birthday[] = {
    {"john",      1,  2},
    {"henry",    27,  3},
    {"sue",       7,  4},
    {"maryanne", 10, 11}
    };
```

is turned into

```
    struct {
    char *name;
```

```
    int day;
    int month;
    }      birthday[] = {

    {
        "john", 1, 2
    },
    {
        "henry", 27, 3
    },
    {
        "sue", 7, 4
    },
    {
        "maryanne", 10, 11
    }
    };
```

which looks horrible and goes out of its way to put the variable name in a bad place.

```
> auto-indenting editors are very valuable,
```

I do not call tools that systematically destroy valuable layout clues for the sake of enforcing half a style blindly "valuable".

```
> as the fact that the
> indentation goes wrong is an immediate clue you've made a
mistake.
```

I repeat an earlier observation: this ONLY works if you are writing a SINGLE language in a source file and you are NOT using a literate programming tool. And my experience with Emacs in several languages has been that the indentation going wrong is almost always an artefact of Emacs getting it wrong and has nothing to do with any errors of mine.

Typically, after I've been editing Prolog for about half an hour, I'll use my editor's built-in Prolog syntax check to look for typos. This finds the mistakes that auto-indentation might have found (but probably wouldn't) *and* a lot more (such as singleton variable errors). One great thing about this is that I can limit the check to the part of a buffer that *is* Prolog. I can check Prolog clauses embedded in

LaTeX documents or embedded in C comments as easily as I can check Prolog in files that are nothing but Prolog.

```
>> The problem with comments like % TBD is that they are
still there
>> 4 years and 7 releases later.  tbd/1 and list_undefined/0
are excellent
>
> True, but if the problem never became serious who cares?
```

Someone trying to maintain the program. How do you *know* the problem never became serious? Maybe the people with the problem just gave up.

Right now I am reading a book about a rather exciting area of statistics. It was formatted in LaTeX, and *all* the chapter references in the text are "chapter TEX. NAME" (where TEX.NAME is some internal tag in capitals and maybe some punctuation) instead of to chapter numbers. It's a real pain. (Just in case it's not obvious, the internal tags are not the same as nor are they abbreviations of the chapter titles). This is a published hardback book.

```
>> (albeit SWI-specific) advice.
>
> Or any other system with a cross-referencer.
```

The name of the command is SWI-specific. That's what I meant.

From: **Jan Wielemaker**
Subject: Re: Good Examples of Properly Commented Prolog Code

Instead of filling this forum, it would be more valuable to enhance the original style guide ... I've sent a message to Michael who doesn't appear to be on the mailinglist anymore (or he changed address).

Richard A. O'Keefe wrote:

```
> I've been using Lisp long enough to find the name
'unwind_protect'
> more immediately comprehensible than 'call_cleanup', but
never mind.
> Actually, no.  We SHOULD mind.  Because if we use the name
```

```
> 'unwind_protect' there is an immediately obvious name for
the
> subordinate predicate:
```

call_cleanup/2 was introduced by SICStus (at least thats where I found it first). I prefer to keep the name compatible.

```
>>load_file(File) :-
>>    open(File, read, In),
>>    unwind_protect(protected_load_stream(In), close(In)).
>
> Here there are two changes to the name:
>
> (1) load_<<file>> becomes load_<<stream>>, because the one
is given a
```

I already suggested that for this case

```
> (2) <<>>load_stream becomes <<protected_>>load_stream to
make it clear
>     that this should only be called in the scope of an
unwind_protect/2
>     that will do whatever cleanup is necessary.
```

Actually this isn't correct. You're not going to call write/2 write_protected/2, I may hope. load_stream/1 is a perfectly valid predicate and like all I/O predicates working on streams doesn't close the stream itself and is capable of generating errors. Actually almost any predicate is capable of generating resource errors, and almost all code needs to be protected at some level against this possibility.

_protected or whatever is generally a good idea for with_mutex(+Mutex, :Goal), in which case it is generally not allowed to call the `protected' code directly and this is not immediately obvious.

```
>>or writing your own,
>
> I've tried that too.  The thing that I found when I did it
is that
> RIGID INDENTATION STYLES ARE WRONG.  For example, it often
pays to
> lay things out in a tabular style.  Auto-indenters destroy
```

```
that.
> In fact, that's one of the standing complaints about indent
(1).
```

Indent is good if you receive code written in a style you don't like at all and wish to reuse/examine/... it. The result is generally not perfect, but at least more easy to read than the original. Syntax support in an editor is different as, while writing, you have the option to overrule the editor. Indeed, quite a few of the style rules have exceptions.

```
>>auto-indenting editors are very valuable,
>
> I do not call tools that systematically destroy valuable
layout clues
> for the sake of enforcing half a style blindly "valuable".
```

It it was `half a style' I'd agree. For Prolog I follow PceEmacs probably about 99% of the time. Ok, I wrote it myself ...

```
>>as the fact that the
>>indentation goes wrong is an immediate clue you've made a
mistake.
>
> I repeat an earlier observation:  this ONLY works if you
are writing a
> SINGLE language in a source file and you are NOT using a
literate
> programming tool.  And my experience with Emacs in several
languages has
> been that the indentation going wrong is almost always an
artefact of Emacs
> getting it wrong and has nothing to do with any errors of
mine.
```

Emacs Prolog mode is pretty poor. Its written by a Lisp programmer :-) Its not all that difficult to train an editor to recognise multiple contexts in one file. Of course this only applies to a literal programming system. The few times I write Prolog code in C files I can live without indentation support or switch to Prolog mode and back.

```
>>> The problem with comments like % TBD is that they are
```

```
still there
>>> 4 years and 7 releases later.  tbd/1 and
list_undefined/0 are excellent
>
>> True, but if the problem never became serious who cares?
>
> Someone trying to maintain the program.
```

He'll generally be more happy with a TBD remark, indicating where and what the original author thought something should have been done, than nothing at all.

```
> How do you *know* the problem never became serious?
> Maybe the people with the problem just gave up.
```

Thats possible. SWI-Prolog and all its libraries is full of TBD issues. Most of them never hurt. Quite more often there are issues not flagged as TBD that cause people to react :-)

From: **Jan Wielemaker**
Subject: Re: Good Examples of Properly Commented Prolog Code

Jan Wielemaker wrote:

```
> Instead of filling this forum, it would be more valuable
to enhance the
> original style guide ...  I've sent a message to Michael
who doesn't
> appear to be on the mailinglist anymore (or he changed
address).
```

I've received the LaTeX source from Michael. We are allowed to modify it, as long as he remains author of course. One plan could be to incorporate the results of the discussion, add SWI-Prolog specific concerns and add it as a chapter to the SWI-Prolog manual. Other plans? Someone willing to invest some time?

From: **Roberto Bagnara**
Subject: Re: Good Examples of Properly Commented Prolog Code

Jan Wielemaker wrote:

```
> On Thursday 16 March 2006 17:33, Jan Wielemaker wrote:
>> Instead of filling this forum, it would be more valuable
to enhance the
>> original style guide ...  I've sent a message to Michael
who doesn't
>> appear to be on the mailinglist anymore (or he changed
address).
>
> I've received the LaTeX source from Michael. We are
allowed to modify
> it, as long as he remains author of course. One plan could
be to
> incorporate the results of the discussion, add SWI-Prolog
specific
> concerns and add it as a chapter to the SWI-Prolog
manual.  Other
> plans?  Someone willing to invest some time?
```

I am. Actually, my original plan was to write a document that could serve as a basis for the current development efforts of my group. I would have submitted that document to this list for further comments. Of course, I would have included some of "my personal habits". However, another possibility is to first improve Michael's manuscript trying to be as objective as possible: the issues of personal taste can always be added at a later stage and the intermediate document can be useful to the community.

From: **Bart Demoen**
Subject: Re:Good Examples of Properly Commented Prolog Code

```
> as a chapter to the SWI-Prolog manual.  Other
> plans?  Someone willing to invest some time?
```

I would just suggest that one doesn't present "my personal habits" as general advice to others. There was a lot of that in the recent "discussion".

From: **Jan Wielemaker**
Subject: Re: Good Examples of Properly Commented Prolog Code

Bart Demoen wrote:

```
>> as a chapter to the SWI-Prolog manual.  Other
>> plans?  Someone willing to invest some time?
>
> I would just suggest that one doesn't present "my personal
habits" as
> general advice to others. There was a lot of that in the
recent
> "discussion".
```

I do not agree. Ok, coding style has a personal element. I do see a lot of poor style and I'm pretty sure it will help if there is good document to point to that explains what (a) good style is. Even well written code that is not in _your_ style is generally hard to read, which make it desirable to share one style in a community. I think a large majortity of the statements from Michael's document and the remarks is not just a matter of personal taste.

From: **Simon Price**
Subject: Re: Good Examples of Properly Commented Prolog Code

Guidelines are always a mixture of "what's good for everyone" and "what's good for me" but the hope is that there will be more convergence on the personal habits side so that it is easier to read/maintain each others' code. In many cases it doesn't matter what The Style is so long as people use it. The Java community, for example, has benefited from a well thought out (mostly) style document that came out with the language. Editors do have a part to play but are much more a matter of personal choice or an accident of personal history. Personally I really appreciate the sharing of ideas on Prolog style - even the habits.

From: **Wlngg**
Subject: Re: Good Examples of Properly Commented Prolog Code

... Just to mention one dimension which has been neglected so far in the discussion: the non-English programming world.

With the arrival of Unicode, programming languages (e.g. Java), following "global" applications (e.g. Word), have made sometimes considerable efforts to allow localisation.

Could or Should SWI-Prolog follow in the footsteps of Java? May be not right now. That SWI-Prolog allows now the possibility of programming on the basis of the Unicode encoding is in itself a great improvement. The terse syntax of Prolog should only make the main task of adapting the programming environment to different cultural contexts easier.

Stylistic considerations, while dependent on the cultural context, are very important for communications purposes. What I would like to see more of is the increased possibility for programming languages to be customisable to the specific needs of the cultural and linguistic contexts of the programming community, or alternatively, to be generic enough to cater for different cultural and linguistic contexts (the second alternative being the one I personally favour). For example, LPA Prolog and even Strawberry Prolog allow predicate names in "any" script (I have tried it with the first one).

Prolog has a great advantage over languages like Java or C ..., through its minimalist syntax. Except for built-ins and the few reserved words, the rest is already pretty international. This should allow for greater flexibility in Prolog.

To illustrate my point about the need for improvement, take the case of variables' notation in Prolog : the rule that variables start with uppercase letters among other (PrologIII takes lower case letters for variables and upper-case for constants, but a primed upper-case like C' becomes a variable, etc.) is good for languages that have the lower/uppercase differentiation; many scripts don't have such a differentiation. We could do with a more "global" or generic syntax for variables, and for other aspects of the Prolog language, ... after all, Prolog is about programming "in logic".

From: **Richard O'Keefe**
Subject: Re: Good Examples of Properly Commented Prolog Code

Simon Price wrote:
```
> Guidelines are always a mixture of "what's good for
everyone"
> and "what's good for me" but the hope is that there will
be more
> convergence on the personal habits side so that it is
easier to
> read/maintain each others' code.
```

The Ada Quality and Style Guidelines are a shining model of how to do it well. One example of that concerns identifier casing: Ada is not case sensitive, so tom_paine, Tom_Paine, TOM_PAINE, toM_painE and so on all mean the same. The first edition of the AQ&S guidelines had one recommendation. Current editions have a *different* recommendation, with the remark that experience proved the original recommendation didn't work as well as they hoped.

It is possible to present style guidelines in a "patterns" sort of way:

- What is the problem?
- What tradeoffs might we have to make?
- What are some alternatives?
- What is the recommendation?

In this case, the specific recommendation might have a "what's good for me" element, but we may with some reason hope that the rest of the material remains useful even for people who think they have a better answer.

```
> In many cases it doesn't matter what The Style is so long
as
> people use it.  The Java community, for example, has
benefited
> from a well thought out (mostly) style document that came
out
> with the language.
```

Well, no, it *didn't* come out with the language. One of my students at RMIT developed a style guide for Java *before* the official one came out. (Versions of Java *were* available before 1.0.)

From: **Richard O'Keefe**
Subject: Re: Good Examples of Properly Commented Prolog Code

Wlngg raises the issue of internationalisation.

1. Quintus Prolog allowed "any script" back in the late-80s. My proposal to the ISO Committee dealt with this clearly and in some detail; like most of my proposals it was completely ignored.
2. I don't know what SWI Prolog does with unquoted identifiers, but it can certainly handle any characters in quoted atoms.
3. Concerning variables, the solution which was adopted back in the mid-80s was ever so simple: for scripts that do not have (or, in the case of Georgian, do not use) a lower-case/upper case distinction, variables just begin with "_". This means that the singleton variable check has to be modified slightly:

    - do NOT warn about a variable that only occurs once if it begins with "_" followed by an upper case letter or if it begins with "__" followed by a non-case letter
    - DO warn about a variable that occurs more than once if it begins with "_" followed by an upper case letter or if it begins with "__" followed by a non-case letter

    So _Fred is a named singleton, _(Hanzi) is an ordinary variable where (Hanzi) is a Chinese character, and __(Hanzi) is a named singleton using a Chinese character.
    This preserves the rule "stick an underscore in front of a variable name to keep it out of singleton warnings".

Like I said, this is a problem that was solved back in the late 80s.

From: **Wlngg**
Subject: Re: Good Examples of Properly Commented Prolog Code

Richard O'Keefe replied:

```
> (1) Quintus Prolog allowed "any script" back in the late-
80s.
>     My proposal to the ISO Committee dealt with this
clearly and
>     in some detail; like most of my proposals it was
completely ignored.
```

With Unicode available now (contrary to the 80s), your proposals should no longer remain ignored. For some of us they are of an urgent necessity. In the draft document "An Elementary Prolog Library" (pllib.htm) Richard O'Keefe wrote the following statement: "In order to deal effectively with Unicode (and it was the plain responsibility of the ISO Prolog committee to address this), some other means entirely will have to be found." If that is still the case, then SWI-Prolog could provide some of those means without compromising Prolog's integrity. I think the key principle is that of genericity: if the satisfaction of a new and specific requirement leads to a greater genericity of the underlying Prolog syntax, then it should be upheld

I could list a dozen other individual and specific requirements Prolog could satisfy: programmers with SOV (Subject Object Verb) natural languages, would wish to be able to write the predicate

```
    is_father_of(abraham,isaac) ...
```

as

```
    (abraham,isaac)father_of_is ...
```

further more, in their own script.

Whether or not these new requirements compromise the integrity of Prolog or entail peformance degradations, it is for the language designers and compiler writers to decide and accommodate.

In any case, I believe that greater syntactic genericity is the way ahead for Prolog (or any other language for that matter.)

From: **Jan Wielemaker**
Subject: Re: Good Examples of Properly Commented Prolog Code

Richard A. O'Keefe wrote:

```
> (2) I don't know what SWI Prolog does with unquoted
identifiers,
>     but it can certainly handle any characters in quoted
atoms.
```

It follows the rules of Prolog, assuming there are no `singletons' outside the ASCII range. So, an unquoted atom is a lowercase followed by a sequence of letter|digit or a sequence of punctuation characters. A variable is uppercase or _ followed by letter|digit. The classification is left to iswupper(), etc. of the C-library. Thats not good, mainly because C libraries differ seriously in how they implement iswupper() and friends. Notably some only provide meaningful results for the `installed languages', while others provide support regardless of installed languages for a smaller or larger subset of the unicode range. The big problem is that it depends on the C-library and/or installed languages whether a program can be loaded on another machine.

So, I think I should include tables in Prolog that do the classications we need and use that in the parser. Its on my todo list. It isn't a very big issue and it is on my TODO list. Would be good to have testers willing to provide active feedback, preferable with some knowledge on coding issues.

From: **Bart Demoen**
Subject: Re: Good Examples of Properly Commented Prolog Code

```
> programmers with SOV (Subject Object Verb) natural
languages, would=
>
> wish to be able to write the predicate
>
>    is_father_of(abraham,isaac) ...
>
> as
>
>    (abraham,isaac)father_of_is ...
```

If you define father_of_is as a postfix operator, you have already (part of) what you want.

But I realise this is an unsatisfactory answer - I myself hate answers that tell me that "I can do it myself" because I am of course as lazy as any other person :-)

So I will give you an even more unsatisfactory answer ...

```
> In any case, I believe that greater syntactic genericity
is the way
> ahead for Prolog (or any other language for that matter.)
```

In the 80-ies, several languages tried the road of "redefinable syntax" or "user-definable syntax". My first dabbling with Prolog implementation involved exactly that. That road didn't become the main language design road - I am not sure why exactly, but the more flexibility in the syntax, the more difficult it is to let others read your program. Prolog still allows you to (re)define the operators - which can make a program totally unreadable to anyone not familiar with your declarations. Mercury is more rigid in that respect. Cobol was an attempt at a syntax closer to natural language - that part of Cobol wasn't that much a success.

If you really want to program in a different syntax, but with an X-like semantics (X being Prolog at this point I presume), I would say that you should do your preprocessor yourself, and not try to bend the X-syntax: there are just too many potential syntax bending ways.

From: **Richard O'Keefe**
Subject: Re: Good Examples of Properly Commented Prolog Code

I wrote:

```
> (1) Quintus Prolog allowed "any script" back in the late-
80s.
>     My proposal to the ISO Committee dealt with this
clearly and
>     in some detail; like most of my proposals it was
completely ignored.
```

Wlngg wrote:

```
> With Unicode available now (contrary to the 80s), your
proposals should no
> longer remain ignored. For some of us they are of an
```

```
urgent necessity.
```

In the 1980s we did not have Unicode, true. But we DID have

- The XNS character set on the Xerox Lisp Machines. (The XNS character set was a 16-bit character set devised by Xerox for Network Services; it included Latin, Greek, Cyrillic, and Japanese characters.)
- Shift-JIS (a quite popular encoding for Japanese)

And there was the ISO 2022 framework as well. So we DID have the problem of dealing with large character sets (we had a Japanese terminal and tested the Japanese support in-house). What we DIDN'T have was a *single* large character set to support.

In fact, one of the earliest documents in the catalogue of documents distributed to the standardisers, PS/6, already talked about the need to support large character sets back in 1984 (or maybe '83; I don't remember when I wrote it).

```
> I could list a dozen other individual and specific
requirements
> Prolog could satisfy: programmers with SOV (Subject Object
Verb)
> natural languages, would wish to be able to write the
predicate
>
>    is_father_of(abraham,isaac) ...
>
> as
>    (abraham,isaac)father_of_is ...
>
> further more, in their own script.
```

I note that Prolog's normal order, Verb Subject Object, does NOT match the typology of the languages spoken by Prolog's designers (which are SVO).

There is no question that programmers should be able to use their own script. I firmly believe that programmers should be able to write numbers in their own script as well as identifiers (my lexical proposal to the ISO committee allowed this). In fact, I see no reason why I shouldn't be allowed to write 2(1/2) using the Latin-1 (1/2) character.

As for their own syntax, I don't think it is reasonable to ask a standard to support every variation (all six permutations of SVO, pro-drop or non-pro-drop, switch reference, ...) that the world's natural languages exhibit. But there _is_ a case to be made for allowing people to plug in their own parser. One standard syntax plus one standard way to plug in your own parser means that someone who wants (x,y)p can write a parser for that once.

Pluggable parsers for the compiler would of course have no *runtime* performance implications.

From: **Wlngg**
Subject: Re: Good Examples of Properly Commented Prolog Code

Bart Demoen wrote:

```
> If you define father_of_is as a postfix operator, you have
already
> (part of) what you want.
```

... is the xyf mode possible in SWI-Prolog?

```
> In the 80-ies, several languages tried the road of
"redefinable
> syntax" or "user-definable syntax". My first dabbling with
Prolog
> implementation involved exactly that. That road didn't
become the main
> language design road - I am not sure why exactly, but the
more
> flexibility in the syntax, the more difficult it is to let
others read
> your program. Prolog still allows you to (re)define the
operators -
> which can make a program totally unreadable to anyone not
familiar
> with your declarations.
```

Well, in the 80s many people were (rightly) happy with 7-bit encoding. It was also

in the 70s-80s that we had the "Edinburgh"/"Marseilles" Prologs: PrologIII still offers both styles. Unicode brings with it many more challenges that will certainly be to the great advantage of Prolog: its syntax should, ultimately cater for most needs, with the possibility of reverting any specific form back to the "standard" form (as it has been possible in the past, to switch from the "Marseilles" to the "Edinburgh" Prolog and vice-versa). The latter could be considered an aberration; whereas the first is a necessity: the Arabic programmers would use a right-to-left style in their own script, as well as the Hebrew programmers, etc. Each programming community should have no difficulty communicating among itself, or with the wider Prolog community. What matters is that they are all programming "in Prolog". Having said that, I do appreciate the complexities involved, and the required changes in attitude. All in good time ... For now what is really urgently required is the ability to program in Prolog, with the Latin as well as the non-Latin scripts.

Richard O'Keefe wrote:

```
> There is no question that programmers should be able to
use their
> own script. I firmly believe that programmers should be
able to
> write numbers in their own script as well as identifiers
(my lexical
> proposal to the ISO committee allowed this). [...] As for
their own
> syntax, I don't think it is reasonable to ask a standard
to support
> every variation (all six permutations of SVO, pro-drop or
> non-pro-drop, switch reference, ...) that the world's
natural
> languages exhibit. But there _is_ a case to be made for
allowing
> people to plug in their own parser.
```

Thanks Richard and Bart. I leave you with a greater satisfaction that Prolog is in good hands!

From: **Roberto Bagnara**

Subject: Re: Good Examples of Properly Commented Prolog Code

Jan Wielemaker wrote:

> Bart Demoen wrote:
>>> as a chapter to the SWI-Prolog manual. Other
>>> plans? Someone willing to invest some time?
>> I would just suggest that one doesn't present "my personal habits" as
>> general advice to others. There was a lot of that in the recent
>> "discussion".
>
> I do not agree. Ok, coding style has a personal element. I do see a lot
> of poor style and I'm pretty sure it will help if there is good document
> to point to that explains what (a) good style is. Even well written code
> that is not in _your_ style is generally hard to read, which make it
> desirable to share one style in a community. I think a large majortity
> of the statements from Michael's document and the remarks is not just a
> matter of personal taste.

Simon Price wrote:

 > Guidelines are always a mixture of "what's good for everyone" and
 > "what's good for me" but the hope is that there will be more convergence
 > on the personal habits side so that it is easier to read/maintain each
 > others' code. In many cases it doesn't matter what The Style is so long
 > as people use it. The Java community, for example, has benefited from a
 > well thought out (mostly) style document that came out with the
 > language. Editors do have a part to play but are much

```
more a matter of
 > personal choice or an accident of personal history.
Personally I really
 > appreciate the sharing of ideas on Prolog style - even
the habits.
```

I agree with Jan and Simon. Moreover, as long as the personal habits come along with their rationale, they can be weighted against each other and all will help to improve the state of things (which is, as Jan says, dominated by poor style, starting from my own sources).

Concerning coding guidelines that could have an impact on the community, I think they should present both the basic and the more sophisticate techniques. For example, it is clear that

```
<name>(<mode><var>: <type>, ..., <mode><var>: <type>)
```

is a very good way to begin the documentation of a predicate. Especially if one uses the extended set of modes indicated by Richard (instead of the simpler and less expressive {+, -, ?}). And it is even better if <type> is specified in a formal way, perhaps in a way that enables the use of automatic type checkers... However, since the perfect is the enemy of the good, I believe we should not put things this way. It is probably more productive to start with a "basic level" where types can be omitted and where the mode system is the simpler one and then explain why the more complex mode system is strictly superior and why specifying types is beneficial. I believe this way it would be easier to have convergence on the basic level. If we make sure the more advanced levels are extensions of the more basic ones and we succeed in explaining why the more advanced techniques are worth their higher cost, then I believe many people will adopt the basic level (which is already a nice thing) and, being conscious of its limitations, some of them will gradually move to more advanced levels. People working on big project may instead decide to enforce the more advanced guidelines from the beginning.

From: **Paulo Moura**
Subject: Re: Good Examples of Properly Commented Prolog Code

Richard A. O'Keefe wrote:

```
>     = (not in Covington)    not necessarily ground, but
not further bound
>     so for example it is simply wrong to write
>     compare(?R: order, +T1: term, +T2: term)
>     because T1 and T2 are allowed to be variables, and
>     compare(?R: order, ?T1: term, ?T2: term)
>     is misleading because we expect that ? arguments will
be unified
>     with something, but
>     compare(?R: order, =T1: term, =T2: term)
>     is just right.
```

The Prolog ISO standard (Part I) uses "@" with the same meaning (section 8.1.2.2; page 64). Thus, compare/3 would be specified as:

```
    compare(?order, @term, @term)
```

From: **Richard O'Keefe**
Subject: Re: Good Examples of Properly Commented Prolog Code

Paulo Moura wrote about "extended" modes:

```
> The Prolog ISO standard (Part I) uses "@" with the same
meaning
> (section 8.1.2.2; page 64). Thus, compare/3 would be
specified as:
>
>     compare(?order, @term, @term)
```

I keep the ISO Prolog substandard on-line, but I never expect to find any good ideas in it. The "@" notation has an obvious relationship to the term comparison predicates @< and so on. That makes '@' arguably more intention-revealing than '='. Noted.

From: **Roberto Bagnara**
Subject: Re: Good Examples of Properly Commented Prolog Code

Richard A. O'Keefe wrote:

```
> Steve Moyle wrote:
>> One place to start would be Covington's guidlines:
>>
>> [PDF] Some Coding Guidelines for Prolog
>> http://www.ai.uga.edu/mc/plcoding.pdf
>
> I have a few quibbles with that.
>
> [...]
>
> 6.6 Test every predicate by forcing it to backtrack.
>
>     Misleadingly worded:  many predicates *can't*
backtrack.
>     "Test every predicate by failing back into it"; now
that's
>     something you *can* do.
```

I would go a bit further here, since the verb "test" alone does not mean much. I would give the explicit advice of checking that failing back into the predicate _really__does_ (as apposed to _apparently_does_) the right thing. For example, especially (but not only) when the predicate at hand is meant to be deterministic or semideterministic, checking that choicepoints left around are as expected is often the source of interesting surprises. I believe the paragraph about the importance of mastering the debugger should explicitly mention this: I met several people (and I am not talking only about students) that were completely unsuspecting of the possibility of obtaining a list of pending alternatives.

```
> 6.7 Test predicates by supplying arguments of the wrong
types.
>
>     I'm not so sure about this one.  For a beginner, it
may be useful
>     advice so that they find out what is likely to provoke
which
>     error message.  But append/3, for example, is designed
```

```
to work
>     with lists; it isn't *intended* to do anything in
particular if
>     given arguments of any other type, so there is no
sense in which
>     you can test it for other types.  (To test something,
there must
>     be a specified set of acceptable behaviours so that
you can tell
>     whether the actual behaviour is acceptable or not.  If
"anything
>     goes", then the "test" cannot fail.)
```

Perhaps it means: "When a predicate is part of a public interface, make sure it fails straightaway when arguments do not satisfy the requirements set by the interface (instead behaving in a bizarre and completely unpredictable way." This advice is good for any language. But I believe for a language such as Prolog is particularly good: systematic checking of the interfaces has saved me lot of headaches. My 2 cents, of course.

Noted.

From: **Paul Singleton**
Subject: Re: Good Examples of Properly Commented Prolog Code

Roberto Bagnara wrote:

```
>> 6.6 Test every predicate by forcing it to backtrack.

> ...I would give the explicit advice of checking that
> failing back into the predicate _really__does_ (as apposed
to
> _apparently_does_) the right thing.  For example,
especially (but
> not only) when the predicate at hand is meant to be
deterministic
> or semideterministic, checking that choicepoints left
around are as
```

```
> expected is often the source of interesting surprises.
```

Is there a reliable programmatic way of checking whether a goal leaves a choicepoint? e.g.

```
succeeds_without_choicepoint(Goal) :-
    statistics(localused, L1),
    call(Goal),
    statistics(localused, L2),
    L1 >= L2.

?- succeeds_without_choicepoint(member(1, [1,2,3])).

No
?- succeeds_without_choicepoint(memberchk(1, [1,2,3])).

Yes
```

but I guess all sorts of things could go wrong with this?

Nested Predicates?
From: **Maurizio Colucci**

Whenever I realize that a predicate is only used inside another one, I miss the ability to nest predicates. It would make the code more readable: instead of seeing a flat space, the reader would only see a few top-level predicates, and make his way into the code in a hierarchical way, like traversing a tree.

Is there any plan for such an addition to SWI? Is there any standard? Is there any counter-argument or alternative solution? Thanks

From: **Paulo Moura**
Subject: Re: Nested Predicates?

You may use SWI-Prolog module system or Logtalk objects to encapsulate auxiliary predicates and expose only a "few top-level predicates" as you write above.

From: **Jan Wielemaker**
To: SWI-Prolog mailing list
Subject: Re: [SWIPL] Nested predicates?
Date: Sun, 26 Mar 2006 17:44:26 +0100

Maurizio Colucci wrote:

```
> Whenever I realize that a predicate is only used inside
another one, I
> miss the ability to nest predicates. It would make the
code more
> readable: instead of seeing a flat space, the reader would
only see a
> few top-level predicates, and make his way into the code
in a
> hierarchical way, like traversing a tree.
>
> Is there any plan for such an addition to SWI? Is there
any standard?
```

I think the answer is 'no no'.

```
> Is there any counter-argument or alternative solution?
Thanks
```

As suggested, modules (or objects) are the standard way around. As for counter arguments, I think is needlessly complicates syntax and would require various new primitives to examine the program, do meta-interpretation, etc. If there is something to be nested, modules come to mind as the first candidate. The currently flat module space is getting a problem as the amount of reusable code grows, especially if people aren't careful to use some kind of prefix to avoid name-clashes.

From: **Richard A. O'Keefe**
To: SWI-Prolog mailing list
Subject: Re: [SWIPL] Nested predicates?
Date: Mon, 27 Mar 2006 15:34:55 +1200 (NZST)

Maurizio Colucci wrote:

```
> Whenever I realize that a predicate is only used inside
another one,
> I miss the ability to nest predicates.
```

When I started writing Prolog code at Edinburgh, I used to indent auxiliary predicates. Taking the example of calculating Fibonacci numbers, which happened to come up in another mailing list, I would have written

```
    :- mode fib(+, ?).


    fib(0, 1) :- !.
    fib(N, F) :-
    integer(N), N > 0,
    fib(N, 1, 1, S),
    F = S.

    :- mode fib(+, +, +, -).

    fib(1, X, _, X) :- !.
    fib(N, X, Y, S) :-
        N1 is N - 1,
        W is X + Y,
        fib(N1, W, X, S).
```

Lawrence Byrd looked at some of my code, and said "You really miss Algol, don't you?"

```
> It would make the code more readable:
```

Once I got used to the fact that Prolog really isn't that kind of language, I realised that no, it *didn't* make the code more readable.

```
> instead of seeing a flat space, the reader would only see a
> few top-level predicates, and make his way into the code
in a
```

> hierarchical way, like traversing a tree.

This is one of the reasons. Things are like a tree much less often than one thinks. An auxiliary predicate having been defined, you often find uses for it elsewhere. (For example, my fib/3 can also be used to calculate Lucas numbers.)

> Is there any plan for such an addition to SWI?

I hope not.

> Is there any standard?

Yes. The standard is "don't do that".

> Is there any counter-argument or alternative solution?

The readability of your code depends far more on the quality of your comments than on nesting. I don't really believe that there is any problem here that needs a solution. The way to encapsulation stuff is to use modules.

Oddly enough, I've recently been suggesting child modules for another similar language in another mailing list. Child modules would be nice for very large systems. Curiously enough, I know of two languages that *had* hierarchical modules (SETL and Lisp) which dropped them (ISETL and SETL2 have a flat module name space, as does Common Lisp).

From: **Richard A. O'Keefe**
To: SWI-Prolog mailing list
Subject: RE: [SWIPL] Nested predicates?
Date: Tue, 28 Mar 2006 14:07:09 +1200 (NZST)

Ralf Lammel wrote, rather strangely,
> I don't see any convincing argument in Richard O'Keefe's email that
> explains why "nested predicates" in Prolog are *conceptually* less
> sensible than local function definitions in Algol, Haskell, Pascal, ...

That's because I didn't even *try* to provide such an argument, and that's because

I did not make any such claim.

I did not say, sing, whistle, hum, inscribe on clay, stone, sand, wax, or any other material, transmit by Morse code, mental telepathy, pheromones, or any other means, nor cause nor induce any other person to do so, any statement which could be held as meaning or implying that "nested predicates are conceptually less sensible in Prolog".

They aren't *necessary*.
They would greatly complicate the language.
They would greatly increase the difficulty of writing tools to process the language.
Using Prolog since October 1979 has shown me that they would have VERY low payoff.

But none of that says that they are "conceptually less sensible".

Of *course* you could have a logic programming language with nested predicates, and if you want one, by all means go ahead and design and implement it. Just don't expect anyone else to pay the price.

```
> Richard says that "Things are like a tree much less often
than
> one thinks."  but how does this explain that local
definitions
> are used quite a bit in say Haskell
```

As it happens, between the time I wrote this sentence and the time I'll write the next one, I'm going to give a 2 hour lecture on Haskell programming [left at 10:54am]. [returned at 1:06pm] Can I make the obvious point that "Things are like a tree much less often than one thinks" was a statement about Prolog, so cannot fairly be expected to explain anything about Haskell?

Let me also make the obvious point that Haskell *needs* local definitions because it doesn't have unification. Consider the following Haskell code for computing the "percentage bend correlation":

```
pbcor = pbcor_gen 0.1

pbcor_gen beta pairs =
    dot as bs / sqrt (dot as as * dot bs bs)
    where
    dot xs ys = sum [x*y | (x,y) <- zip xs ys]
```

```
    as = scaled_and_clamped xs
        bs = scaled_and_clamped ys
        (xs, ys) = unzip pairs

        scaled_and_clamped xs =
            [(0-1) `max` (1 `min` ((x-phi)/omega)) | x <- xs]
            where
        phi = (omega*fromIntegral (i2-i1) + s)/fromIntegral
(n-i1-i2)
                s = sum ((n-i1-i2) `take` (i1 `drop`
sorted_xs))
                sorted_xs = sort xs
        i1 = length [1 | x <- xs, (x-median)/omega < -1]
        i2 = length [1 | x <- xs, (x-median)/omega > 1]
                omega = sort ws !! (floor ((1 - beta)
*fromIntegral n) - 1)
                ws = [abs (x - median) | x <- xs]
                median = (sorted_xs !! h + sorted_xs !! (n-1-
h)) / 2
                h = n `div` 2
                n = length xs
```

In this Haskell code, there are
2 nested functions
14 nested variables
5 list comprehensions

What does a Haskell compiler do with the nested functions? IT MOVES THEM OUT! (This is called 'lambda lifting', IIRC.) We can do the same for Prolog.

```
pbcor(Pairs, R) :-
    pbcor(Pairs, 0.1, R).

pbcor(Pairs, Beta, R) :-
    unzip(Pairs, Xs, Ys),        % no let/where is needed for
    scaled_and_clamped(Xs, Beta, As),   % introducing Xs,
Ys, As, or Bs.
    scaled_and_clapmed(Ys, Beta, Bs),   % (nor AA, AB, or
BB).
    dot(As, As, AA),
    dot(As, Bs, AB),
```

```
    dot(Bs, Bs, BB),
    R is AB/sqrt(AA*BB).

scaled_and_clamped(Xs, Beta, Ys) :-
    length(Xs, N),            % once again, there is no need
    H1 is N // 2,            % for any let/where just so that
    H2 is N - 1 - H1,            % we can define local
variables!
    msort(Xs, Sorted_Xs),
    nth0(H1, Sorted_Xs, Mid_Lo),
    nth0(H2, Sorted_Xs, Mid_Hi),
    Median is (Mid_Lo + Mid_Hi)/2,
    Median_Shift is -Median,
    scaled_and_shifted(Xs, 1, Median_Shift, Ms),
    abs_list(Ms, Ws),
    msort(Ws, Sorted_Ws),
    Omega_Pos is floor((1 - Beta)*N) - 1,
    nth0(Omega_Pos, Sorted_Ws, Omega),
    count(X, Ms, X < -Omega, I1),
    count(X, Ms, X >  Omega, I2),
    I3 is N - I1 - I2,
    drop(I1, Sorted_Xs, Mid_And_High_Xs),
    take(I3, Mid_And_High_Xs, Mid_Xs),
    sum(Mid_Xs, S),
    Phi is (Omega*(I2-I1) + S)/I3,
    Scale is 1.0/Omega,
    Shift is -Phi/Omega,
    scale_and_shift(Xs, Scale, Shift, Scaled_And_Shifted),
    clamp(Scaled_And_Shifted, -1, 1, Ys).

/*  Library predicates  */
/*  All of these are or should be in your Prolog library,
    possibly with a different name and interface,
    except for scale_and_shift/4, clamp/4, abs_list/2.
    I'm a bit embarrassed about those, to tell you the
    truth, but NONE of these functions is coupled to its
    uses in the preceding code; ALL are reusable.
*/
unzip([], [], []).
unzip([(X,Y)|Pairs], [X|Xs], [Y|Ys]) :-
    unzip(Pairs, Xs, Ys).
```

```
dot(Xs, Ys, Dot) :-
    dot(Xs, Ys, 0, Dot).

%  NB: at first sight, dot/4 should be private to dot/3,
%  but I have *VERY* often had a use for calling it directly
%  instead of calling dot/3.

dot([], [], Dot, Dot).
dot([X|Xs], [Y|Ys], Dot0, Dot) :-
    Dot1 is Dot0 + X*Y,
    dot(Xs, Ys, Dot1, Dot).

sum(Xs, Sum) :-
    sum(Xs, 0, Sum).

%  NB: at first sight, sum/3 should be private to sum/2,
%  but again, I find it useful to call sum/3 directly about
%  as often as it's useful to call sum/2.

sum([], Sum, Sum).
sum([X|Xs], Sum0, Sum) :-
    Sum1 is Sum0 + X,
    sum(Xs, Sum1, Sum).

scale_and_shift([], _, _, []).
scale_and_shift([X|Xs], A, B, [Y|Ys]) :-
    Y is A*X+B,
    scale_and_sfhit(Xs, A, B, Ys).

clamp([], _, _, []).
clamp([X|Xs], L, U, [Y|Ys]) :-
    (   X < L -> Y = L
    ;   X > U -> Y = U
    ;            Y = X
    ),
    clamp(Xs, L, U, Ys).

count(Template, List, Condition, Count) :-
    findall(*, ( member(Template, List), Condition ), L),
    length(L, Count).
```

```
abs_list([], []).
abs_list([X|Xs], [Y|Ys]) :-
    Y is abs(X),
    abs_list(Xs, Ys).

drop(N, [_|Xs], Ys) :- N > 0, !,
    N1 is N - 1,
    drop(N1, Xs, Ys).
drop(_, Xs, Xs).

take(N, [X|Xs], [X|Ys]) :- N > 0, !,
    N1 is N - 1,
    take(N1, Xs, Ys).
take(0, _, []) :- !.
take(_, [], []).

nth0(N, List, X) :-
    (   integer(N) ->
    N >= 0,
    nth0i(N, List, X)
    ;   var(X) ->
    nth0v(L, X, 0, N)
    ;   abort
    ).

nth0v([X|_], X, N, N).
nth0v([_|Xs], X, N0, N) :-
    N1 is N0 + 1,
    nth0v(Xs, X, N1, N).

nth0i(N, {X|Xs], V) :-
    (   N =:= 0 -> V = X
    ;   N1 is N - 1,
        nth0i(N1, Xs, V)
    ).
```

Oh, let's clean that up a bit. The preferred way to take a section of a list in Prolog is not to use take and drop but to use length:sublist.

```
    drop(I1, Sorted_Xs, Mid_And_High_Xs),
    take(I3, Mid_And_High_Xs, Mid_Xs),
```

should be

```
    length:sublist(Sorted_Xs, Mid_Xs, I1, I3, I2)
```

We can do list comprehensions with findall/3:

```
    Median_Shift is -Median,
    scaled_and_shifted(Xs, 1, Median_Shift, Ms),
    abs_list(Ms, Ws),
```

could be

```
    findall(W, (member(X, Xs), W is abs(X-Median)), Ws)
```

and

```
    Scale is 1.0/Omega,
    Shift is -Phi/Omega,
    scale_and_shift(Xs, Scale, Shift, Scaled_And_Shifted),
    clamp(Scaled_And_Shifted, -1, 1, Ys).
```

could be

```
    findall(Y, ( member(X, Xs), Y is min(1, max(-1, (X-Phi)/
Omega)) ), Ys)
```

I just wish findall/3 were as cheap as "map", but it isn't.

```
> but they wouldn't be used in Prolog, if they were enabled?
```

I dare say they WOULD be used in Prolog. *BUT* "enabling" nested definitions in Prolog would not be a simple matter of flipping a switch (as "enabled" suggests), but of fairly major design and implementation work. And the compiler would have to implement them the way a Haskell compiler does: by turning them into UN-nested functions.

```
> I could try to guess that this
> may have something to do with the more pervasive higher-
order
> style in Haskell, but I would think that people also use
> "where"s in first-order Haskell programs.
```

Of course they do. They use 'where' a LOT in Haskell, for the simple reason that if you want to name an intermediate result (that is NOT a function) you have to use "let" or "where".

```
> I don't have metrics results that back up my guess.  I
could
> come up with some other guesses; I have indeed some
candidates,
> but perhaps someone *knows*.
```

Take a look at Mercury. It combines logic programming and functional programming, and it is quite good at supporting higher order programming, and it HAS anonymous nested function and predicate definitions (tamed by the mode system so that higher-order unification is never needed).

In short, if you don't like Prolog, try Mercury. It may be exactly what you want. (And it requires a very complex implementation, but someone has already done that.)

```
> In fact, the (first-order) fib function appears to me as a
reasonable
> example of a function that benefits from a local helper.
Richard says
> "An auxiliary predicate having been defined, you often
find uses for it
> elsewhere. (For example, my fib/3 can also be used to
calculate Lucas
> numbers.)" I get the point for fib/3 but the general
argument seems to
> lead to a slippery slope because it sounds a bit like
let's presume most
> abstractions are reusable in perhaps unanticipated ways,
so better
> expose them, so better export almost everything.
```

No, I did *NOT* say "export". In fact I said the explicit opposite of that. I said to ENCAPSULATE functions that would have been nested, but to encapsulate them using MODULES.

In fact the example I presented above is a good (because entirely honest) demonstration of my point: with the single exception of scaled_and_clamped/3,

every single auxiliary predicate I needed either *was* in my library already or could profitably be put there. Ones that weren't already in the library would not be *EXPORTED*; they would be kept at top level until they were needed in another module at which point they would be moved out of this module entirely and into an appropriate module in the library.

```
> Below, I list some ways to transcribe O'Keefe's fib to
Haskell. The
> first one is the more sensible one for the case of fib; it
associates
> the helper with the relevant equation. The other two
variations
> illustrate notational options -- one can make it so that a
helper is
> accessible by several "cases". In reality, one uses local
functions for
> more than just "hiding" or "grouping with a client", i.e.,
they are used
> as means to take advantage of the existing argument
bindings for the
> parent. The fib example cannot make interesting use of
this possibility
> but here is a lambda-dropped version of append that binds
ys at the top:
>
> append xs ys = append xs
>  where
>   append [] = ys
>   append (x:xs) = x : append xs
```

And do you know what a Haskell compiler does with this? It turns it into

```
    append xs ys = (append' ys) xs

    append' ys [] = ys
    append' ys (x:xs) = x : (append' ys) xs
```

which is just the original definition twisted around a bit. One has to ask, what *is* the point? Why move something in when it's going to be moved straight out again?

In fact there is a specific feature of Haskell which means that nested functions are often better as un-nested functions (with export being controlled in Haskell, as in SWI Prolog, by the module system). That's the well known scope-of-type-variables problem.

```
> -- Attach the helper to the relevant equation
> fib 0 = 1
> fib n = fib n 1 1
>  where fib 1 x _ = x
>        fib n x y = fib (n-1) (x+y) x
```

Which the Haskell compiler turns into

```
    fib 0 = 1
    fib n = fib' n 1 1

    fib' 1 x _ = x
    fib' n x y = fib' (n-1) (x+y) x
```

The nested function is HARDER to read because the human reader has to ask "WHY is this function nested? HOW is it coupled to its surroundings?" In this case, the answer turns out to be "it isn't", which causes a justified feeling of resentment: "why, Mr Author, did you make me do all that decoding work for nothing?"

```
> -- Factor for a single binding, multiple RHSs
> fib n | n == 0    = 1
>       | otherwise = fib n 1 1
>  where fib 1 x _ = x
>        fib n x y = fib (n-1) (x+y) x
```

This has all the disadvantages of the previous version, plus some scoping weirdness: 'where' definitions are available in multiple alternatives for a single rule, but not for multiple rules. Again, this is far more readable if the nested function is NOT nested.

Just to try, possibly vainly, to avoid any misconception: I am NOT saying that nested functions are undesirable or inappropriate or anything like that in Haskell. What I *am* saying is that if an auxiliary function isn't *coupled* to its context through one or more shared variables, it is almost always better to un-nest it. (I wrote the pbcor function at the top, and the only reason dot is declared inside is sheer laziness; I am sorry I did that.)

From: **Bart Demoen**
To: SWI-Prolog mailing list
Subject: Re: [SWIPL] Nested predicates?
Date: Tue, 28 Mar 2006 21:47:39 +0200

Richard A. O'Keefe wrote:

```
> if an auxiliary function isn't *coupled* to its context
through one
> or more shared variables, it is almost always better to un-
nest it.
> ...
> the only reason dot is declared inside is sheer laziness
```

[I didn't read all of Richard's mail]

I find the ability to nest functions/predicates attractive because I can often vouch for "this definition is correct when used as in the context just surrounding it" but not for any larger context. That's lazyness, or lack of time, but most of all it is trying to err on the safe side: don't make things public that you're not prepared to "maintain".

"*coupled* to its context through one or more shared variables" is one aspect of coupling to a context - it couls also be because of preconditions that are context dependent.

From: **Richard A. O'Keefe**
To: SWI-Prolog mailing list
Subject: Re: [SWIPL] Nested predicates?
Date: Wed, 29 Mar 2006 16:51:25 +1200 (NZST)

I wrote:

```
>> if an auxiliary function isn't *coupled* to its context
through one
>> or more shared variables, it is almost always better to
un-nest it.
```

Bart Demoen replied:

```
> I find the ability to nest functions/predicates attractive
because I
> can often vouch for "this definition is correct when used
as in the
> context just surrounding it" but not for any larger
context.
```

This brings us back to the discussion about style that we were having a while back. What is to the advantage of the *writer* of a piece of code is not always to the advantage of the *reader*.

To me the important question is "how will someone ELSE, reading this code, KNOW what the relevant context actually is?"

```
> That's lazyness, or lack of time, but most of all it is
trying to err
> on the safe side: don't make things public that you're not
prepared to
> "maintain".
```

This is a straw man. Consider this table:

	public	**not public**
nested	possible	possible
not nested	possible	possible

I have been, and remain, concerned solely with the nested/not nested question. This is entirely orthogonal to the public/not public question. All four combinations are theoretically possible and all four of them are supported in some programming language or other. (Lisp, for instance.)

Making something "not nested" is very very VERY different from making it public.

```
> "*coupled* to its context through one or more shared
variables" is one
> aspect of coupling to a context - it could also be because
of
> preconditions that are context dependent.
```

Right. Which is why simply putting one routine inside another one is NOT enough

all by itself. A future reader (including yourself a few months later) needs to know WHAT the relevant context is. That means commenting it.

Now, take an example from my earlier message:

```
    nth0(N, List, X) :-
    (   integer(N) ->
        N >= 0,
        nth0i(N, List, X)
    ;   var(X) ->
        nth0v(L, X, 0, N)
    ;   abort
    ).

    nth0v([X|_], X, N, N).
    nth0v([_|Xs], X, N0, N) :-
    N1 is N0 + 1,
    nth0v(Xs, X, N1, N).

    nth0i(N, {X|Xs], V) :-
    (   N =:= 0 -> V = X
    ;   N1 is N - 1,
        nth0i(N1, Xs, V)
    ).
```

nth0/3 is exported from the module it is defined in. It is "public" and "not nested".

nth0v/4 and nth0i/3 are *not* exported from that module. They are "not public" and "not nested".

Making them "not nested" means a big benefit for a human reader: whatever the context might be (if any) that links them to nth0/3, it *isn't* shared variables. When I am trying to understand a variable in nth0i/3, I do not have to look anywhere else at all.

There _is_ a coupling between these predicates and their caller, and in the real code, it's explained in the comments I stripped out to keep the previous message short:

```
    %.  nth0v(?List: list(T), ?Element: T, +N0: integer, -N:
integer).
    %.  nth0i(?List: list(T), ?Element: T, +I: integer).
```

So, would this be better if we made these predicates nested?

```
nth0(N, List, X) :-
(   integer(N) ->
    N >= 0,
    nth0i(N, List, X)
    where {|
    nth0i(N, {X|Xs], V) :-
        (   N =:= 0 -> V = X
        ;   N1 is N - 1,
        nth0i(N1, Xs, V)
        )
    |}
;   var(X) ->
    nth0v(L, X, 0, N)
    where {|
    nth0v([X|_], X, N, N);;
    nth0v([_|Xs], X, N0, N) :-
        N1 is N0 + 1,
        nth0v(Xs, X, N1, N)
    |}
;   abort
).
```

I don't think so. We run into a prlblem: are N and X the same thing inside nth0i/3 as they are in nth0/3, and if not, why not? The sheer bulk makes this version of nth0/3 hard to read.

But most tellingly, there is *also* in that library a predicate

```
nth1(N, L, X) :-
    (   integer(N) ->
        N >= 1,
        M is N - 1,
        nth0i(M, L, X)
    ;   var(N) ->
        nth0v(L, X, 1, N)
    ;   abort
    ).
```

Yes, those predicates depended on instantiation state guaranteed by the caller, BUT another caller within the same module could make the same guarantees.

If routines are unnested ("lambda-lifted") and the context conditions are made EXPLICIT, then

- the routines, though still private, are available for further use within the same module
- whether they are used again or not, the life of human readers is made vastly simpler because they are TOLD what they would otherwise have to puzzle out.

Make no mistake. I have known and loved Algol-like languages for a long time. (The second non-trivial program I ever worked on was a port of Wirth's Euler compiler.) I have known and loved Lisp-like languages for a long time. I regard the lack of nesting functions in C++ as a serious flaw. Give me a language that allows nested definitions, and I will use them.

I'm not arguing that nested definitions are a bad idea.
I'm making two claims:

1. *implicit* coupling is not good.
2. One of the main advantages of Prolog is its simplicity; adding nested definitions to Prolog would make it a much more complicated language, and empirically, the benefit would not be great.

As I've said before, if I want Mercury, I know where to find it. (~/export/mercury.d/, as it happens.)

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

Archive of the ALP Newsletter

- The archive at http://www.cs.kuleuven.ac.be/~dtai/projects/ALP/newsletter/archive_93_96/archive.html covers Vol. 6/1, February 1993 to Vol. 10/2, May 1996.
- Between May 1996 and May 2001 no newsletters have been archived.
- Vol 14 n. 2, May 2001
- Vol 14 n. 3, August 2001
- Vol 14 n. 4, November 2001
- Vol 15 n. 1, February 2002
- Vol 15 n. 2, May 2002
- Vol 15 n. 3, August 2002
- Vol 15 n. 4, November 2002
- Vol 16 n. 1, February 2003
- Vol 16 n. 2 and 3, May and August 2003
- Vol 16 n. 4, November 2003
- Vol 17 n. 1, February 2004
- Vol 17 n. 2, May 2004
- Vol 17 n. 3, August 2004
- Vol 17 n. 4, November 2004
- Vol. 18 n. 1, February 2005
- Vol. 18 n. 2, May 2005
- Vol. 18 n. 3, August 2005
- Vol. 18 n. 4, November 2005
- Vol. 19 n. 1, February 2006
- Vol. 19 n. 2, May 2006
- Vol. 19 n. 3, August 2006

Enrico Pontelli