Cooperation of the Finite Domain and Set Solvers in TOY

S. Estévez-Martín, A. J. Fernández, and F. Sáenz-Pérez

Abstract

This paper describes the cooperation mechanism between two constraint domains, namely Finite Domain and Set, that has been implemented in the TOY system. The former one was already described in previous works, whereas the latter is a new addition to TOY, which is a novelty presented here for the first time. This work introduces the Set solver, shows its components (data values, functions, and operators), describes a number of details about its implementation, and also explains the way in which the two mentioned domains cooperate. In addition, the paper provides a number of examples that highlight the functionalities of the Set solver and also illustrate the feasibility of our cooperation proposal.

Keywords: Constraints, Solver Cooperation, Finite Domains, Sets, Functional Logic Programming

1 Introduction

Nowadays, applications arise in highly heterogeneous scenarios requiring the integration of several software technologies. Satisfactory and optimization applications are revealed as critical examples for which different data structures and types are needed to be combined. On the one hand, there exists the need to integrate these different data domains, whilst, on the other, specific solving techniques are demanded to compute in each application domain. High-level programming systems help to develop such applications by abstractions, as the use of constraints. Examples of such systems lay in the declarative programming paradigm.

Here, we focus on the constraint functional logic programming (CFLP) system TOY [1], a state-of-the-art representative of this paradigm, which embodies several constraint domains and allows their integration and cooperation. It implements the cooperation mechanisms presented in [3]. In this paper, we propose the integration

1 First and third authors partially supported by projects TIN2006-03419-C04-01, TIN2006-06622-C04-01, S-0505/TIC/04917 and CCMICE-101-0490622 - second author partially supported by Spanish MICINN under contract TIN2008-05941 (NEMESIS project)
of the new constraint domain \( S \) which implements data types, functions, and operators on polymorphic sets. This constraint domain is based on interval lattices \([6,7]\) and, in addition, allows the communication and cooperation between the finite domain and set constraint domains is proposed. An example supporting the applicability of this integration is presented.

2 Related Work

Most of the existing constraint solvers provide support for solving on specified domains; this implies that the constraints are usually restricted to just values in the given computation domain. However, in practice, many problems require heterogeneous constraints expressed using more than one domain and, thus, problems representations have to be artificially adapted to fit the particular choice of domain and solver. A solution to this problem has been found in the concept of solver cooperation, an issue that has attracted the attention of many researchers in the constraint programming (CP) community. Here we mention \([2,10,12,13]\) as a limited selection of references illustrating various approaches to the problem.

In this paper we consider the cooperation between the finite and set domains. The finite domain is very well known in CP due to its practicality \([11]\), whereas \( S \) is another important domain employed traditionally in the constraint logic programming (CLP) setting because sets enable the modelling of problems formulated on combinatorial solving and on natural language processing. In the literature one can find a number of proposals to integrate sets in constraint solvers \([9,13,14,16]\).

3 An Overview of TOY\((S)\)

The set constraint solver \( S \) (solver \( S \)) used in the TOY system has been implemented as a particular instance of the schema described in \([6,7]\). This schema, based on interval lattices, is a generic framework for both interval constraint satisfaction and interval solver cooperation, and defines a constraint solving proposal for CLP\((\text{Interval}(\Sigma))\), i.e., CLP on intervals defined on any computation domain \( X \) with lattice structure. Here \( X \) is the lattice \( S_{\Delta} \), i.e., the domain of sets defined on \( \Delta \), i.e., \( X \) is the lattice \( S_{\Delta} \), i.e., the domain of sets defined on \( \Delta \), where \( \Delta \) is a lattice where \( \subseteq \), \( \cup \) and \( \cap \) define respectively a partial order on the computation domain, the join or least upper bound, and the meet or greatest lower bound. The bottom element of this lattice is the empty set \( \emptyset \) and the top element is the set containing all the elements of \( \Delta \).

This section is devoted to describe the TOY\((S)\) solver. We show the main components of our solver, illustrate how constraint solving is defined, and describe briefly its implementation. A number of simple examples that highlight the functionalities of this solver are also provided in order to help the reader to understand the set interval constraint solving process integrated into the TOY system.
Each function \( f \) has an associated principal type of the form \( \tau_1 \rightarrow \cdots \rightarrow \tau_n \rightarrow \tau \). As usual in functional programming, types are inferred and, optionally, can be declared in the program. Defining rules for a function \( f \) have the basic form \( f \colon \tau_1 \times \cdots \times \tau_n \to \tau \). Informally, the intended meaning of a program rule is that a call to \( f \) can be narrowed to \( \tau \) whenever the actual parameters match the patterns \( \tau_i \) and the conditions in \( \Gamma \) are satisfied. An n-ary predicate is viewed as a particular kind of function, with type \( \tau_1 \to \cdots \to \tau_n \to \text{bool} \). As a syntactic facility, we can use \( \text{clauses} \) as a shorthand for defining rules whose right-hand side is true. This allows one to write Prolog-like predicate definitions: each clause \( p_1 \ldots p_n \leftarrow C_1, \ldots, C_k \) abbreviates a defining rule of the form \( p_1 \ldots p_n = \text{true} \leftrightarrow C_1, \ldots, C_k \).

### 1.3 Set Constraints in TOQ\(S\)

The available set constraints and functions we define as follows, where \( \text{int} \), \( \text{bool} \), and \( \text{setOfInt} \) denote the type for integers, Booleans, and set of integers, resp. This last type can be read as the datatype declaration \( \text{data setOfInt} = \text{set} \text{[int]} \).

- \( \text{domainSet} \{ S_1, \ldots, S_n \} \{ \{ \{ X_1, \ldots, X_k \} \{ Y_1, \ldots, Y_l \} \} \{ \{ \text{setOfInt} \text{[int]} \rightarrow \text{setOfInt} \rightarrow \text{setOfInt} \rightarrow \text{bool} \} \} \}

  imposes the following conjunction of set interval constraints:

  \[ \bigwedge_{1 \leq i \leq n} S_i \subseteq \{ X_1, \ldots, X_k \} \}

• \( \text{subset} \ S_1 \ S_2 \{ \text{with type} \text{setOfInt} \rightarrow \text{setOfInt} \rightarrow \text{bool} \} \}

  imposes that each element of the set \( S_1 \) belongs to the set \( S_2 \), reducing the domain of \( S_1 \) if necessary. Analogously for \( \text{superSet} \).

• \( \text{intersectSet} \ S_1 \ S_2 \{ \text{with type} \text{setOfInt} \rightarrow \text{setOfInt} \rightarrow \text{setOfInt} \} \}

  returns the set intersection of \( S_1 \) and \( S_2 \).

• \( \text{unionSet} \ S_1 \ S_2 \{ \text{with type} \text{setOfInt} \rightarrow \text{setOfInt} \rightarrow \text{setOfInt} \} \)

  returns the set union of \( S_1 \) and \( S_2 \).

• \( \text{subtractSet} \ S_1 \ S_2 \{ \text{with type} \text{setOfInt} \rightarrow \text{setOfInt} \rightarrow \text{setOfInt} \} \)

  returns a set with the elements in \( S_1 \) that do not occur in \( S_2 \).

• \( \text{removeFromSet} \ A \ S \{ \text{with type} \text{int} \rightarrow \text{setOfInt} \rightarrow \text{setOfInt} \} \)

  returns a set with the elements in \( S \) but the element \( A \).

• \( \text{cardinalSet} \ S \{ \text{with type} \text{setOfInt} \rightarrow \text{int} \} \)

  returns the cardinality of \( S \).

Regarding \( \text{labelingSet} / 1 \), we note that this function is defined in order to provide a complete \( \text{TOQ}(S) \) solver as it is well-known that, in general, propagation algorithms are not enough to solve a constraint satisfaction problem (CSP) and, as a consequence, it is very frequent that, when no more constraint propagation is possible, an additional labeling strategy is employed to solve the CSP. This process is very usual in association with solvers over finite and discrete domains.

### 3.4 Inference Rules

In this subsection we show (cf. Table 1, where \( S_i \), \( 1 \leq i \leq 3 \), denotes a set variable, \( F \) a finite domain variable, \( \# \) the cardinality of a set \( S \), and, as usual, \( A \) the set difference) a number of inference rules that guide the process of set interval constraint solving in \( \text{TOQ}(S) \). Constraint solving is executed as a combination of constraint propagation, constraint narrowing, and variable enumeration. Both constraint narrowing and enumeration of set variables are a direct consequence of applying their corresponding inference rules; constraint propagation is produced via the application of the rest of rules.

#### Table 1

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Inference rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>( S_i[a] = S_j ) \rightarrow S_i[a] = S_j</td>
</tr>
<tr>
<td>Inconsistency</td>
<td>( S_I \neq S_j ) \rightarrow S_I \neq S_j</td>
</tr>
<tr>
<td>Subset</td>
<td>( S_3 \subseteq S_2, S_2 \subseteq S_1 \rightarrow S_3 \subseteq S_1 )</td>
</tr>
<tr>
<td>SuperSet</td>
<td>( S_1 \subseteq S_3, S_3 \subseteq S_2 \rightarrow S_1 \subseteq S_2 )</td>
</tr>
<tr>
<td>Union</td>
<td>( S_1 \cup S_2 = S_3, S_2 \cup S_1 = S_3 \rightarrow S_1 \cup S_2 = S_3 )</td>
</tr>
<tr>
<td>Intersection</td>
<td>( S_1 \cap S_2 = S_3, S_2 \cap S_1 = S_3 \rightarrow S_1 \cap S_2 = S_3 )</td>
</tr>
<tr>
<td>Subtract</td>
<td>( S_1 \setminus S_2 = S_3, S_3 \setminus S_2 = S_1 \rightarrow S_1 \setminus S_2 = S_3 )</td>
</tr>
<tr>
<td>CardinalSet</td>
<td>( S \times \text{cardinalSet} \ S F \rightarrow (S \times \text{cardinalSet} \ S F) )</td>
</tr>
<tr>
<td>Constraint</td>
<td>( S_i[a] = S_j[a] ) \rightarrow S_i[a] = S_j[a]</td>
</tr>
<tr>
<td>Narrowing (CH)</td>
<td>( S_i[a] = S_j[a] ) \rightarrow S_i[a] = S_j[a]</td>
</tr>
<tr>
<td>Set labeling</td>
<td>( S_i[a] = S_j[a] ) \rightarrow S_i[a] = S_j[a]</td>
</tr>
</tbody>
</table>

Some Inference Rules for the Set Interval Constraint Solver
Example 3.3 In the following we illustrate a process of constraint solving (without labeling) where inference rules have been applied in an arbitrary order. ⇒6 denotes the rule 6 applied in each step, and the selected \( TOC(S) \) constraints for this rule in the inference process are underlined. Note that, opposed to usual deductions, we show some intermediate steps which are performed by the constraint solver and not described by the inference rules. In particular, by applying the Subset rule, only one deduction step should appear. Here, we add conclusions without removing hypotheses but when pruning domain by merging them via the CN rule:

\[
\begin{align*}
S_1 \subseteq \{1\}, \{1,2,3\} & \quad S_2 \subseteq \{1\}, \{1,2,3,4\} & \quad S_1 \subseteq S_2 & \Rightarrow 6 \\
S_1 \subseteq \{1\}, \{1,2,3\} & \quad S_2 \subseteq \{1\}, \{1,2,3,4\} & \quad S_1 \subseteq S_2 & \Rightarrow 7 \\
S_1 \subseteq \{1\}, \{1,2,3\} & \quad S_2 \subseteq \{1\}, \{1,2,3,4\} & \quad S_1 \subseteq S_2 & \Rightarrow 8 \\
S_1 \subseteq \{1\}, \{1,2,3\} & \quad S_2 \subseteq \{1\}, \{1,2,3,4\} & \quad S_1 \subseteq S_2 & \Rightarrow 9
\end{align*}
\]

Note that some rules might have been combined to obtain further domain reduction. For instance, CN and Subset rules can be combined as follows:

\[
\begin{align*}
S_1 & \subseteq \{a, b\} & S_2 & \subseteq \{c, d\} & S_1 & \subseteq S_2 \\
S_1 & \subseteq \{a, b\} & S_2 & \subseteq \{c \cup a, d\}
\end{align*}
\]

\( TOC(S) \) has been implemented in SICStus Prolog via Constraint Handling Rules (CHR)'s \( \mu \); these rules are a proposal to allow more flexibility and application-oriented customization of constraint systems. In this paper we will neither discuss optimizations nor further implementation issues as this is a topic of further work.

4 Cooperation between the Domains \( S \) and \( FD \)

In this section, we explain the cooperation among the pure constraint domains \( S \), just described, and \( FD \), which supplies arithmetic and finite domain constraints over integers (see \[1,4\]). Table 2 recalls some primitive functions.

Our cooperative computation model keeps different stores for constraints corresponding to different domains. In addition, there is a special store, called mediator store (\( MFS \)) where the communication constraints between the finite domain and the set domain are placed. Communication constraints (\( F \rightharpoonup S \), following the type in Table 2) are called bridges and can be used to project constraints involving the variable \( S \) into constraints involving the variable \( F \), as we will see in Section 4.1.

4.1 Projections from \( S \) to \( FD \)

Projection takes place whenever a constraint is submitted to the \( S \) solver. At that moment, projection rules relying on the available cooperation constraints (\( \rightharpoonup \)) are used for building mate constraints which are submitted to the \( FD \) solver. Table 3 shows the available opportunities for projections, assuming there exist bridges \( F \rightharpoonup S \).

The second column of this table shows new bridges calculated in the computing process, where the symbol means that no bridges are created. The third column shows the \( FD \) constraints which are projected, where they correspond to the constraints listed in the \( TOC \) user manual [1], but \( setcomplement F1 \ F2 \) (constrains
prunes the domain of F3 to the feasible values 10, 20.

Now, if the subset S2 S1 constraint is added, then the domain of the variable S2 is reduced to \( \{10, 20\} \), and the corresponding projection reduces the domain of the variable F2 to \( \{10, 20\} \).

### 4.2 A Cooperation Problem

We consider a program written in TCO for solving the problem of scheduling tasks that require resources to complete, and have to fulfill precedence constraints. Figure 1 shows a precedence graph for six tasks which are labelled as \( T_{X,Y} \), where \( X \) stands for the identifier of a task \( L \), \( Y \) for its time to complete (duration), and \( Z \) for the identifier of a machine \( m \) (a resource needed for performing the task \( X \)).

![Fig. 1 Precedence Graph](image)

The following program models this scheduling problem.

1. \( \text{durationList} :: [\text{int}] \)
   \( \text{durationList} = [3, 8, 6, 3, 4] \)

2. \( \text{listFromTo} :: \text{int} -> [\text{int}] \)
   \( \text{listFromTo} X = \text{take} X (\text{iterate} (+ 1) 1) \)

3. \( \text{scheduling} :: [\text{setOfInt}] -> [\text{int}] -> \text{bool} \)
   \( \text{scheduling} \text{TasksSet} \text{TasksFD} = \text{true} \)

4. \( \text{TasksSet} = [T1S, T2S, T3S, T4S, T5S, T6S] \)
5. \( \text{TasksFD} = [11F, 51F, 31F, 41F, 52F, 72F] \)
6. \( \text{foldl} \text{true} \text{(zipWith} (+) \text{TasksFD TasksSet}) \)
7. \( \text{map} \text{cardinalSet TasksSet = durationList} \)
8. \( \text{X precedences} \)
   \( \text{fd}_{\text{max}} \text{T1F} < \text{fd}_{\text{min}} \text{T6F}, \)
   \( \text{fd}_{\text{max}} \text{T2F} < \text{fd}_{\text{min}} \text{T5F} \)
9. \( \text{X machine} S1 \)
   \( \text{intersectSet T1S T2S (set [1])} \)
   \( \text{intersectSet T3S T4S (set [1])} \)

A task is modelled as a variable of type setOfInt. The time of execution of all tasks is, at most, the sum of the durations of all the tasks, that is, variable \( \text{Time} \) in line (7). Therefore, the time of executing every task can be placed in the time interval defined from 1 to \( \text{Time} \) (line (8)). Note that the empty list [1] means that it is not possible to accomplish this task, and \( [1, 2, 3] \) means that the tasks will be accomplished in the days \( 1, 2, 3 \), where it is not necessary that they are consecutive days. The duration of a task corresponds to the cardinal of its set (line (9)).

The main function \( \text{scheduling} \) takes two arguments: the list of tasks to be scheduled as a list of sets, and the same list of tasks as a list of integer variables. Every set variable is related to a finite domain variable by the cooperation constraint \( \# = \) (line (6)). We use these variables in order to project set constraints into finite domain constraints when possible (cf. Subsection 4.1) and allow the cooperation of both solvers.

The tasks can use certain machines. If two tasks need to use the same machine then they cannot execute at the same time. Therefore, the intersection of the sets that represent them is empty (blocks (11) and (12)).

The precedences among tasks are represented by finite-domain relational operators since this representation is simpler (block (10)).

Figure 2 shows a possible solution for the problem of scheduling tasks.

![Fig. 2 A Possible Solution for the Problem of Scheduling Tasks](image)

Projection is necessary to solve this problem, due to the fact that some constraints work in the \( S \) domain and others in the \( FD \) domain. The projection of the set constraints creates new finite domain constraints which are needed to solve the problem. In particular, the projection of \( \text{domainSet} \) prunes the domain of the finite domain variables to values included in the \( S \) and \( T \) domains. Next, \( \text{cardinalSet} \), because of its projection, limits the number of elements in the domain of the finite domain variable to the number of days (in the list \( \text{durationList} \)). Finally, the \( \text{intersectSet} \) constraints project the finite domain constraints, therefore avoiding overappings.

### 5 Conclusions and Future Work

We have presented a new domain \( S \) (i.e., the set of integers domain) integrated currently in the TCO system which can cooperate with the existing domain \( FD \).
The cooperation is allowed via a new bridge relating S and FD variables. Several constraint functions have been proposed and the S solver has been implemented using CHIPs. We have shown an example of a scheduling problem with a more natural representation, which is in addition solved thanks to the cooperation of the solvers of both domains.

This work can be extended in a number of ways. For instance, we think to renew the current implementation of the set solver with a wider set primitive toolbox; this extension demands obviously the construction of additional bridges relating the set and finite domains. Also, we can study how to redefine the propagation mechanism of the set solver to produce stronger (bound) consistency. In addition, projection from FD to S will be considered as well as alternative models for the FD–S bridges. Moreover, we are working on new examples to motivate the feasibility of our cooperative model.

References


Advances in type systems for Functional-Logic Programming

(Work in progress)

Francisco López-Fraguas1, Enrique Martin-Martín2 and Juan Rodríguez-Hortala3
Deparmento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Madrid, Spain

Abstract

Type systems are widely used in programming languages as a powerful tool to ensure safety to programs, and forcing the programmers to write code in a cleaner way. Functional-Logic languages have inherited Danae & Milner type system from their functional part due to its simplicity and popularity. However, functional-logic languages have some problematic features not taken under consideration by standard systems. In particular, it is known that the use of opaque H0 patterns in left-hand sides of program rules may produce undesirable effects from the point of view of types. We re-examine the problem, and propose a Danae & Milner-like type system whose certain uses of H0 patterns (even opaque) are permitted while preserving type safety, as posed by a subject reduction result that uses a H0 set rewriting, in a recently proposed operational semantics for H0 functional logic programs.

Keywords: Functional logic programming, Type systems, Opaque patterns.

1 Introduction

Type systems for programming languages are an active area of research [18], no matter which paradigm one considers. In the case of functional programming, most type systems have aspired as extensions of Danae & Milner’s [1], for its remarkable simplicity and good properties (decidability, existence of principal types, possibility of type inference). Functional logic languages [11,17], in their practical side, have inherited more or less directly Danae & Milner’s types. In principle, most of the type extensions proposed for functional programming could be also incorporated to functional logic languages (this has been done, for instance, for type classes in Prolog).

* This work has been partially supported by the Spanish projects TIN2005-09267-C03-01, TIN2005-09222-C05-02 and UCS-2006-0044. S-005/TIC/2007 and UCS-2006-0044. The authors are affiliated to COPPE-UFRJ.
1 Email: fraguas@di.ufrj.br
2 Email: martinsn@di.ufrj.br
3 Email: juanrodr@di.ufrj.br

IX Jornadas sobre Programación y Lenguajes
Taller de Programación Funcional
P. Lluch, G. Moreno and R. Peña, Eds.
Santander, 8-11 de septiembre de 2009

226
Programación y Lenguajes

IX Jornadas sobre Programación y Lenguajes. PROLE'09
I Taller de Programación Funcional, TPF'09

San Sebastián, España

del 8 al 11 de Septiembre de 2009

Editores.

PAQUI LUCIO, GINES MORENO Y RICARDO PENA
Comité de Programa de PROLE-2009

Presidente Comité: Ginés Moreno (U. de Castilla-La Mancha)
Elvira Albert (U. Complutense de Madrid)
Jesús Almendros (U. de Almería)
María Alpuente (U. Politécnica de Valencia)
Gilles Barthe (IMDEA)
Miquel Bertran (U. Ramón Llull)
Santiago Escobar (U. Politécnica de Valencia)
Antonio Fernández (U. de Málaga)
Víctor Guillén (U. de A Coruña)
Paqui Lucio (U. del País Vasco)
Narciso Martí-Ollet (U. Complutense de Madrid)
Susana Muñoz (U. Politécnica de Madrid)
Marisa Navarro (U. del País Vasco)
Manuel Niñez (U. Complutense de Madrid)
Fernando Orejas (U. Politécnica de Cataluña)
Yolanda Ortega (U. Complutense de Madrid)
Francisco Ortín (U. de Oviedo)
Ernesto Pimentel (U. de Málaga)
Germán Puebla (U. Politécnica de Madrid)
Enric Rodríguez (U. Politécnica de Cataluña)
Jaime Sánchez (U. Complutense de Madrid)
Germán Vidal (U. Politécnica de Valencia)

Comité de Programa de TPF-2009

Presidente Comité: Ricardo Peña (U. Complutense de Madrid)
Víctor Guillén (U. de A Coruña)
Francisco Gutiérrez (U. de Málaga)
Salvador Lucas (U. Politécnica de Valencia)
Pablo Noguera (U. Politécnica de Madrid)
Julio Rubio (U. de la Rioja)
Alberto Verdejo (U. Complutense de Madrid)
Mateu Villaret (U. de Girona)
Comité Organizador de JISBD/PROLE-2009

Presidenta Comité: Goitria Sagardú (U. de Mondragón)
Vicepresidenta Comité: Paqui Lucio (U. del País Vasco)
Gentzane Aldekoa (U. de Mondragón)
Ana Altuna (U. de Mondragón)
Javier Alvez (U. del País Vasco)
Lorela Belategui (U. de Mondragón)
Leire Etxeberria (U. de Mondragón)
Jose Gaintzarain (U. del País Vasco)
Montserrat Herno (U. del País Vasco)
Marisa Navarro (U. del País Vasco)
Xabier Sagarna (U. de Mondragón)

Comité Ejecutivo de PROLE-2009

Presidente Comité: Fernando Orejas (U. Politécnica de Cataluña)
Jesús Almendros (U. de Almeria)
María Alpuente (U. Politécnica de Valencia)
Manuel Hermenegildo (U. Politécnica de Madrid)
Paqui Lucio (U. del País Vasco)
Juan José Moreno (U. Politécnica de Madrid)
Gines Moreno (U. de Castilla-La Mancha)
Ricardo Peña (U. Complutense de Madrid)
Ernesto Pimentel (U. de Málaga)

Revisores adicionales de PROLE/TPF-2009


Índice

Prólogo ........................................ IX
Charla Invitada ......................................... 1
K. RUSTAN M. LEINO
Understanding program verification ................. 3
Taller de Programación Funcional ................. 5
FRANCISCO-JESÚS MARTÍN-MATEOS, JOSÉ-LUIS RUIZ-REINA,
JULIO RUBIO, LAUREANO LAMBIÁN
Verificación y eficiencia en programas para el cálculo simbólico:
estudio de un caso ................................ 7
MARÍA ALPUENTE, MARCO A. FELÍC, CHRISTOPHE JOUBERT,
ALCIA VILLANCUEVA
Implementing Datalog in Maude ..................... 13
JOSÉ IBORRA
Explicitly Typed Exceptions for Haskell .......... 23
MANUEL MONTEÑEGRO, RICARDO PEÑA, CLARA SIGUERA
Experiences in developing a compiler for Safe using Haskell .......... 31
HENRIQUE FERREIRO, DAVID CASTRO, VÍCTOR M. GULLAS,
AZZE DIJKSTRA
Implementing memory reusing in the UHC Haskell compiler .......... 39
Tipos, Estructuras de Datos y Gestión de Memoria .......... 47
FRANCISCO ORTÍN, DANIEL ZAPIRO
Hacia un sistema de tipos estático y dinámico .......... 49
JAVIER DE DIOS, RICARDO PEÑA, MANUEL MONTEÑEGRO
Certified Absence of Dangling Pointers in a Language with Explicit
Deallocation ........................................ 65
ELVIRA ALBERT, SAMIR GENAIM, MIGUEL GÓMEZ-ZAMALLOA
Live Heap Space Analysis for Languages with Garbage Collection .......... 75
JESÚS MANUEL ALMENDROS JIMÉNEZ
A Rule-based Implementation of XQuery ............. 77
Herramientas y Sistemas Software ............................................. 87
MARINA LLORENS, JAVIER OLIVER, JOSEP SILVA, SALVADOR TAMARIT
An implementation of the MEB and CEB analyses for CSP ........... 89

PASCUAL JULIÁN-IRANZO, CLEMENTE RUBIO-MANZANO
UNICORN: A Programming Environment for Bounsi-Prolog ....... 99

ALEXIS LESCAVELLE, AURICA VILLANUEVA
The teep Interpreter .......................................................... 109

SONIA ESTÉVEZ-MARTÍN, ANTONIO FERNÁNDEZ,
FERNANDO SÁENZ-PÉREZ
TOY: A System for Experimenting with Cooperation of Constraint
Domains ................................................................. 119

SEVIL ALGERI, CRISTINA ZOLTÁN, GUILLERMO PRESTIGIACOMO
NimoToons: a Totally Graphic Workbench for Program Tuning and
Experimentation ......................................................... 129

EVA LINARDE, PABLO ARENAS, SAMIR GENAIR, GERMÁN PUEBLA,
DAMIÁN ZAMARRAIN, DIANA VANESSA RAMÍREZ DE ANTE, MIGUEL
GÓMEZ-ZAMALOA, GUILLERMO ROMÁN-DÍEZ
Termination and Cost Analysis with COSTA and its User Interfaces .. 139

Razonamiento, Lógica y Semánticas ........................................ 149

Mikel ALECHEA, JAVIER ÁLNEZ, MONTSEBRAI HERMO,
EGUZTZ LAPIARRA
A New Proposal for Using First-Order Theorem Provers to Reason with
OWL DL Ontologies ..................................................... 151

GABRIEL ARANDA-LÓPEZ, SUSANA NIEVA, FERNANDO SÁENZ-PÉREZ,
JAIME SÁNCHEZ-HERNÁNDEZ
Implementación de una semántica de punto fijo para un sistema de bases
de datos deductiva con restricciones ................................ 161

FRANCISCO JAVIER LÓPEZ-FRAGUAS, JUAN RODRÍGUEZ-HORTALÁ, JAIME
SÁNCHEZ-HERNÁNDEZ
A Fully Abstract Semantics for Constructor Systems ................. 177

JORDI LEYV, MATEU VILARDET
Nominal Logic from a Higher-Order Perspective ....................... 179

JOSE-LUIS RUIZ-REINA, DAVID A. GREVE, MATT KAFFMANN, PANA-
GIOTES MANOLOS, J. MOORE, SANDIP RAY, ROB SUMMERS, DAREN
VISON, MATTHEW WILDEING
Efficient execution in an automated reasoning environment .......... 181

Programación Lógico Funcional y con Restricciones .................... 183

SONIA SANTIAGO, CAROLYN L. TALCOTT, SANTIAGO ESCOBAR,
CATHERINE MEADOWS, JOSÉ MESEGUER
A Graphical User Interface for Maude-NPA ............................. 185

IGNACIO CASTIñEIRAS, FERNANDO SÁENZ
Integración de ILOG CP en TOY ........................................ 201

SONIA ESTÉVEZ-MARTÍN, ANTONIO FERNÁNDEZ,
FERNANDO SÁENZ-PÉREZ
Cooperation of the Finite Domain and Set Solvers in TOY .......... 217

FRANCISCO JAVIER LÓPEZ-FRAGUAS, ENRIQUE MARTÍN-MARTÍN,
JUAN RODRÍGUEZ-HORTALÁ
Advances in Type Systmes for Functional Logic Programming ........ 227

Análisis de Terminación ..................................................... 237

SALVADOR LUCAS
Automatic proofs of termination with elementary interpretations .... 239

BEATRIZ ALARCÓN, SALVADOR LUCAS, RAFAEL NAVARRO-MARSET
Using Matrix Interpretations over the Reals in Proofs of Termination ... 255

RAUL GUTIÉRREZ, SALVADOR LUCAS
Mechanizing Proofs of Termination in the Context-Sensitive Dependency
Pairs Framework ......................................................... 265

MICHAEL LUECHEL, SALVADOR TAMARIT, GERMAN Vidal
A Fast Procedure for the Strong Termination Analysis of Logic
Programs ........................................................................ 275

CSP, Concurrencia y Poreza .................................................... 285

MARINA LLORENS, JAVIER OLIVER, JOSEP SILVA, SALVADOR TAMARIT
A Semantics for Tracing CSP .............................................. 287

MIQUEL BOFILL, MIQUEL PALAIH, MATEU VILARDET
A system for CSP solving through Satisfiability Modulo Theories . . 303
Prólogo

Las Jornadas sobre PROgramación y Lenguajes (PROLE) se vienen consolidando como un marco propicio de reunión, debate y divulgación para los grupos españoles que investigan en temas relacionados con la programación y los lenguajes de programación. La investigación en este campo está en continuo desarrollo y comprende todo el estudio de conceptos, métodos, técnicas, fundamentos y aplicaciones relativos a la tarea de programar y a los lenguajes que se utilizan en ella. El evento, de carácter anual, pretende fomentar tanto el intercambio de experiencias y resultados, como la comunicación y cooperación entre los grupos de investigadores españoles que trabajan en el área de programación y lenguajes, manteniendo un año más la enriquecedora trayectoria de las ocho ediciones previas celebradas en Almagro (2001), El Escorial (2002), Alicante (2003), Málaga (2004), Granada (2005), Sitges (2006), Zaragoza (2007) y Gijón (2008).

En esta ocasión, la IX edición de las Jornadas (PROLE’09) va precedida por primera vez en su historia del I Taller sobre Programación Funcional (TPF’09). Ambos eventos se celebran entre el 8 y el 11 de septiembre de 2009, dentro de la XXVIII edición de los Cursos de Verano de San Sebastián. Como en ocasiones previas, la organización de esta conferencia se realiza en paralelo con las Jornadas de Ingeniería del Software y Bases de Datos (HSBD’09), compartiendo conferencias invitadas, actos sociales, publicidad, etc. Para información más detallada puede consultarse http://www.mondragon.edu/prole2009/. La organización conjunta de ambos eventos ha sido auspiciada por la Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software (SISTEDS).

Agradece a todos los participantes por la valiosa contribución que han ofrecido, en particular a nuestros oradores invitados. Agradece también a todos los asistentes que han hecho de esta jornada un éxito.

En el ámbito de PROLE’09 se han seleccionado este año un total de 31 trabajos, que cubren tanto aspectos teóricos como prácticos relativos a la especificación, diseño, implementación, análisis y verificación de programas y lenguajes de programación, además de herramientas tangibles y sistemas software que incrementan el carácter pragmático del área. Por su parte, el Taller de Programación Funcional TPF’09 que precede a PROLE’09, inicia su recorrido como una actividad independiente y complementaria a PROLE, con un comité de programa propio que ha seleccionado 5 trabajos recibidos también en esta edición y de investigación en torno a este tipo de lenguajes.

Este volumen recopila por tanto un total de 36 trabajos que fueron rigurosamente revisados cada uno de ellos por 3 miembros de ambos comités de programa y/o revisores adicionales, a los cuales es necesario agradecer su inestimable ayuda y reconocer su gran profesionalidad. También en consonancia...
con este agradecimiento, es justo felicitar a los autores por la calidad de sus trabajos y su contribución a que esta edición sea la de mayor participación en la evolución histórica de PROLE, lo que garantiza la buena salud del evento.

Por otro lado, además de las tres conferencias invitadas que compartimos con la planificación de JISBD'09, el programa de PROLE'09 cuenta este año con una excelente conferencia específica (un resumen de la misma se incluye en este volumen) que, bajo el título de “Understanding program verification”. será impartida por K. Rustan M. Leino, de Microsoft Research, USA, a quien agradecemos el haber aceptado tan amablemente nuestra invitación.

Finalmente, queremos agradecer la confianza que han depositado en nosotros, para conducir la presente edición de estas jornadas, a todos los miembros del comité ejecutivo de PROLE, y esperamos no haberles defraudado. En el desempeño de esta tarea, ha sido determinante la ayuda y experiencia prestada por Jesús Almendros, quien presidió la anterior edición de PROLE en Gijón, y a quien aprovechamos para dar mil gracias desde aquí.

Septiembre de 2009

Pepi Lucio
Gines Moreno
Ricardo Peña

PATROCINADORES