

A Generalised Finite Domain Constraint Solver for SWI-Prolog

Markus Triska, Ulrich Neumerkel, and Jan Wielemaker

Technische Universität Wien, Austria
markus.triska@tuwien.ac.at, ulrich@complang.tuwien.ac.at

Universiteit van Amsterdam, Netherlands
J.Wielemaker@uva.nl

Abstract. In this paper we describe a new constraint solver over finite domains that has recently been included in the SWI-Prolog distribution¹. Our solver generalises finite domain constraint solving towards unbounded domains, and thus enables a uniform approach to integer arithmetic and constraints. We ensure termination of all predicates, which facilitates termination proofs of constraint logic programs. The solver is written in Prolog and can be quite easily ported to other systems.

1 Introduction

Finite domain constraint solvers are typically applied to problems with only quite small values. This is the case in many tasks for which constraint-based approaches are well suited. A well-known benchmark library for constraints, CSPLib ([2]), consists almost exclusively of such examples.

On the other hand, the need for arbitrary precision integer arithmetic is widely recognised, and many common Prolog systems provide transparent built-in support for arbitrarily large integers.

It thus seems natural to enhance a constraint solver over finite domains with the ability to reason over arbitrarily large integers. SICStus Prolog already goes in that direction, using the symbolic constants **inf** and **sup** to denote default domain limits, but internally, they still correspond to quite small integers: The system yields a *representation errors* when these limits are exceeded.

2 Extending domains

We have implemented a new constraint solver over finite domains, in which a software implementation for big integers (*“bignums”*) is transparently used when arising values exceed machine-sized integers. We accept

¹ <http://www.swi-prolog.org>

the **inf/sup** notation of SICStus Prolog, but these atoms now denote the actual infinities instead of abbreviating underlying finite limits.

As an example, consider the so-called “7-11 problem” ([6]), which already surpasses the limits of the finite domain constraint solver of SICStus Prolog on 32-bit systems. This non-linear problem is also beyond the abilities of common CLP(Q) solvers, but can be solved with our solver.

3 Uniform arithmetic

Prolog’s built-in arithmetic predicates are *moded*: At evaluation time, expressions must be ground. Finite domain constraint solvers remove this restriction, but traditionally limit the range of possible values considerably. A significant generalisation was achieved in 1996 with the first release of a finite domain system for SICStus Prolog ([1]): The equality relation $\#=/2$ could now be used in place of $\text{is}/2$ for bignums. In our solver, we generalise this to all constraints, such as $\#</2$ and $\#>/2$, which can be used instead of the built-ins at all places. At compile time, these constraints are specialised to fall back to moded built-in arithmetic for the simply evaluable integer cases. This reduces the overhead for these constraints to about 30% compared to direct usage of built-ins.

4 Ensuring terminating propagation

By allowing unbounded domains we gain significant expressibility, but may receive nonterminating propagation in return. For example, queries like $X\#>\text{abs}(X)$ or $X\#>Y, Y\#>X, X\#>=0$ do not terminate in many existing constraint solvers.

We ensure terminating propagation by allowing the left and right boundaries, as well as the distance between the smallest and largest number occurring in a domain representation to be changed at most once after a constraint is posted, unless the domain is already bounded. That this suffices to guarantee terminating propagation follows from how a non-terminating propagation chain can occur: Either the lower limit of some domain increases, or the upper limit of some domain decreases, or said distance of some domain increases without bound. This weakens propagation for infinite cases, but simplifies termination proofs considerably. In particular, since labeling is now guaranteed to terminate (labeling raises an error with unbounded domains), we obtain a simple criterion for termination of constraint logic programs even with very large search spaces: If posting constraints alone terminates, then this posting followed by a search with labeling also terminates.

5 Implementation

We implemented our constraint solver in Prolog, using attributed variables as described by Demoen ([7]). This interface is provided by (among others) SWI-Prolog ([3]), and our solver is freely available in the latest version of this system as `library(clpfd)`. Our solver can be quite easily ported to other systems that provide attributed variables (as is common), and we also ported the solver to YAP Prolog ([4]).

If all domains are finite, comparisons and computations can be delegated to the Prolog system's built-in support for big integers. For the infinite case, these operations must be generalised appropriately. We did this by implementing a Prolog predicate that behaves like `is/2` and can also handle the symbolic domain boundaries. Similar predicates generalise the built-in comparisons.

We have implemented the common arithmetic constraints (which are also reifiable), and some global constraints like `all_different/1`. The overhead of using generalised predicates for infinities is below 30% for many examples. This agrees with Apt and Zoetewij ([5]), who estimate this overhead to be far less prominent in a full-fledged constraint solver than in isolated examples.

Future work includes performance comparisons with other systems and correctness considerations.

6 Acknowledgments

We thank Tom Schrijvers for his `bounds.pl`, on which our solver was originally based.

References

1. M. Carlsson, G. Ottosson, B. Carlson. An Open-Ended Finite Domain Constraint Solver. PLILP. 1997.
2. I.P. Gent and T. Walsh, CSPLib: A Benchmark Library for Constraints, Proceedings of the 5th Int. Conf. PPCP (1999)
3. Jan Wielemaker, An Overview of the SWI-Prolog Programming Environment, Proceedings of the 13th International Workshop on LP Environments (2003)
4. Anderson Faustino da Silva and Vítor Santos Costa, The Design and Implementation of the YAP Compiler, LNCS 4079 (2006)
5. Krzysztof R. Apt and Peter Zoetewij, An Analysis of Arithmetic Constraints on Integer Intervals, Constraints 4 (2007)
6. Paul Pritchard and David Gries, The Seven-Eleven Problem, TR (1983)
7. Bart Demoen, Dynamic attributes, their hProlog implementation, and a first evaluation, Technical Report (2002)