Towards Automatic Generation and Evaluation of Implied Constraints

Brahim Hnich[1], Julian Richardson[2], and Pierre Flener[3]

[1]Computer Science, Department of Information Science
Uppsala University, Box 513, 751 20 Uppsala, Sweden
Brahim.Hnich@dis.uu.se

[2]Department of Computing and Electrical Engineering
Heriot-Watt University, EH14 4AS Edinburgh, Scotland, UK
julianr@macs.hw.ac.uk*

[3]Computing Science, Department of Information Technology
Uppsala University, Box 337, 751 05 Uppsala, Sweden
Pierre.Flener@it.uu.se

Abstract

Adding appropriate implied constraints can dramatically improve the computational behaviour of a constraint solver on the augmented set of constraints. We briefly document experiments we made with PRESS [10] and clp(q,r) [6] for generating implied constraints.
This report is an excerpt (namely Sections 5.1, 5.2, and 5.3, which are included verbatim apart from minor cosmetic, non-technical corrections) from an unfinished technical report [3] written in August 2000, and thus (finally) provides a citable, published reference to that work.

1 Introduction

In constraint programming, a problem is modelled using a conjunction of formulae, called *constraints*, restricting the values its *decision variables* can take. A *constraint solver* (iteratively) solves these constraints, yielding a solution to the original problem. Constraint solvers are specialised for solving particular kinds of constraints, over particular domains. For example, finite domain solvers handle constraints over variables with finite domains. The constraint programming paradigm allows problems to be specified in a very declarative form, but suffers from a drawback that for many types of constraints it is very hard to predict how long it will

*Current affiliation: Research Institute for Advanced Computer Science, NASA Ames Research Center, MS 269-2, Moffett Field, CA 94035-1000, USA. Email: julianr@riacs.edu

take to solve a problem; one problem may be solved in a matter of seconds while another will fail to be solved in days of runtime.

In this technical report, we explore one way in which computational behaviour of constraint solving can be improved, namely the addition of (non-redundant) *implied constraints*.

Given a set of constraints C over a language $\mathcal{L}$, an *implied constraint* I is a constraint over $\mathcal{L}$ such that $C \cup \{I\} \equiv C$. A *redundant constraint* is an implied constraint that negatively affects the search for a solution, say in time. Adding appropriate (non-redundant) implied constraints can dramatically improve the computational behaviour of a constraint solver on the augmented set of constraints.

In this technical report we briefly document experiments we made with PRESS [10] and clp(q,r) [6] for generating implied constraints. This report is an excerpt (namely Sections 5.1, 5.2, and 5.3, which are included verbatim apart from minor cosmetic, non-technical corrections) from an unfinished technical report [3] written in August 2000, while the first author was a visiting scholar at the Department of Computing and Electrical Engineering, Heriot-Watt University, Scotland. This introduction, and a brief conclusion, have been added in order to supply some context.

2 Generation of Implied Constraints

2.1 Classification of Implied Constraints

We propose the following classification of implied constraints:

- *Simplified implied constraints* (or: *Variable elimination*): Given a constraint C constraining n variables, and a set S of other constraints on these n variables, we try to derive C_{imp} such that $\{C\} \cup S \rightarrow C_{imp}$, and C_{imp} constrains at most $n-1$ of these variables. Here is an example:

$$\begin{array}{c} C \equiv X + Y + Z = 12 \\ S = \{Y > Z\} \\ C_{imp} \equiv X + 2 \cdot Y \geq 12 \end{array} \tag{1}$$

  We call this class C_1.

- *Specialised implied constraints*: We try to derive a specialised implied constraint (often a global constraint), for which there exists a more efficient implementation. Here is an example introducing the *alldifferent* constraint:

$$\begin{array}{c} C \equiv W \neq X \wedge W \neq Y \wedge W \neq Z \wedge X \neq Y \wedge X \neq Z \wedge Y \neq Z \\ C \rightarrow alldifferent(W, X, Y, Z) \end{array} \tag{2}$$

  We call this class C_2.

- *Linear implied constraints inferred from linear and nonlinear constraints*: We try to derive a linear constraint from a set of linear and nonlinear constraints. Here is an example:

$$A^2 = 1 \ mod\ 2 \rightarrow A = 1 \ mod\ 2 \tag{3}$$

  We call this class C_3.

- *Abstract implied constraints* (or: *Variable introduction*): We try to abstract some relationship between some variables of the initial constraint by introducing extra variables and trying to infer implied constraints on them. Here is an example:

$$\begin{array}{c} C \equiv X + Y \neq 2W - 3Z \\ C \rightarrow P = X + Y \wedge Q = 2W - 3Z \wedge P \neq Q \end{array} \tag{4}$$

  We call this class C_4.

- *Abstraction of problems into different constraint systems:* Given a problem formulation in a particular domain (integers, sets, lists, etc), we try to derive some implied constraints on a different domain.

  For example, suppose X, Y, S, T are set variables and, as in Conjunto [2], one can state the constraint $S \cup T = X \wedge S \cap T = Y$. One can now infer the constraints $|S| + |T| \geq |X|$ and $|Y| \leq |S|$, which are in the integer domain.

  We call this class C_5.

- *Implied constraints over a new set of variables*: Given a constraint over a set of variables, we infer a constraint over a subset of these variables for which there is no constraint yet. Here is an example:

$$\begin{array}{c} C \equiv X = Y + Z \wedge W = M - N \wedge Y + Z = 3 \cdot (M - N) \\ C \rightarrow X = 3W \end{array} \tag{5}$$

  We call this class C_6.

2.2 The PRESS System

Overview

PRESS (PRolog Equation Solving System) [10] is a system that automatically solves symbolic equations in one or more variables. It is modular, with different kinds of equation-solving activity defined by *axioms* of the domain, and special-purpose problem-solving procedures, called *methods*, and has in the past been tested on English 'A' level examination questions, which it can mostly solve quite well, with an overall 86.8% success rate. The system generally tries to simplify the (set of) (in)equations it has been asked to solve until it has a solution. PRESS works in the domain of *R-elementary equations*, i.e., equations involving polynomials, as well as exponential, logarithmic, trigonometric, hyperbolic, inverse trigonometric, and inverse hyperbolic functions over real numbers. PRESS has six major methods:

- *Isolation* is a method for solving equations involving only a single occurrence of an unknown. It achieves this by applying some isolation rewrite rules. Examples of such rewrite rules are:

$$\begin{array}{c} log_u V = W \rightarrow V = U^W \\ U - V = W \rightarrow U = V + W \end{array} \tag{6}$$

- *Polysolve* is a collection of polynomial methods that are tried in turn depending on the characteristics of the polynomial, such as being symmetric or anti-symmetric. For example, $x^4 - 4x^2 + 3 = 0$ is identified to be quadratic in x^2 and the solution $x^2 = 1$ or $x^2 = 3$ is found.
- *Collection* is a method to reduce the number of occurrences of an unknown. An example of such a rewrite rule is $(U+V) \cdot (U-V) \rightarrow U^2 - V^2$.
- *Attraction* is a method for bringing the occurrences of an unknown closer. For instance, the rule $log_u V + log_u W \rightarrow log_u(V \cdot W)$ is an attraction rule.
- *Homogenisation* parses the equation and collects the terms that contain the unknown that are non algebraic in the unknown. Then it tries to replace such terms by some algebraic function of some single term. For instance, $(e^x)^3 - 4 \cdot e^x + 3/e^x = 0$ can be changed to $y^3 - 4 \cdot y + 3/y$, where y is substituted for e^x.
- *Function swapping* is a collection of methods that transform an equation according to preference. For instance, $\sqrt{x+y} = 2$ is transformed to $(x+y)^2 = 4$.

Since PRESS is modular, it seemed like a good candidate for our initial experiments in deriving implied constraints.

Improvements to the PRESS System

We modified PRESS is several ways in order to better suit our purposes:[1]

- (Trivially) we updated the system to run on current versions of Prolog (such as SICStus Prolog, version 3 [1]).
- We extensively modified the system to allow it to solve inequalities. Rather than extending the existing inequality module, which is rather minimal, we chose to generalise the equation-solving procedures, for example finding roots of polynomials, to deal with inequalities. A key component of this solution was to extend the notion of substitution from the preexisting substitution using equalities — such as $x = B \vdash \phi \rightarrow \phi[B/x]$ — to inequalities. Care must be taken over the direction of the inequalities and the sign of any coefficients multiplying the substituted variable. The following table describes which substitutions using inequalities can be made, and how the principal connective is effected:

Expression	$L < R$	$L \leq R$	$L > R$	$L \geq R$	$L = R$
Substitution					
$A < B$	$\times$	$\times$	$>$	$>$	$>$
$A \leq B$	$\times$	$\times$	$>$	$\geq$	$\geq$
$A > B$	$<$	$\leq$	$\times$	$\times$	$<$
$A \geq B$	$<$	$\leq$	$\times$	$\times$	$\leq$
$A = B$	$<$	$\leq$	$>$	$\geq$	$=$

[1]The following files were changed, from the distribution version: ./pressdir/toplevel/sim, ./pressdir/toplevel/simeq, ./pressdir/pressjunk/filin, ./pressdir/toplevel/solve, ./pressdir/axioms/ineqis.ax, ./pressdir/methods/poly, and ./util/struct.pl.

The table shows the main connective when an occurrence of A in the left hand side of the formula has been replaced with B. A '$\times$' in a cell means that the substitution cannot be made. Substitution on the right hand side is defined in terms of substitution on the left hand side.

As when rewriting under implication and negation, each position in the expression has a *polarity*. The polarity of the subterm of B in $A - B$ is the opposite of the polarity of the subterm of B in $A + B$. Likewise for $A - 2 \cdot B$ compared to $A + 2 \cdot B$. Substitution of terms that are in a position of negative polarity reverses the inequality of the substituting term. A notion of unification on relations is defined to ensure that the principal connective in a substituting formula is compatible with the main connective in the expression on which the substitution is operating, and what the resulting connective should be.

The following further work immediately suggests itself:

- Provide PRESS with relevant theory for whatever constraint domain we are tackling.
- PRESS tries to solve (in)equations. If it cannot find a solution, it considers that it has failed, and backtracks to the most recent choice point. In fact, although the result of its manipulations may not constitute a "solution" to the problem, it may contain very useful implied constraints. We will therefore modify the system with *acceptance criteria*, which are predicates applied at the leaves of its derivation tree[2] to determine if the derived formula should be kept as a candidate implied constraint. If it is accepted in this way, then it is recorded in Prolog's internal database and retrieved when PRESS has finished its attempt to find a solution.
- PRESS is heavily biased to finding a *solution*. The guidance may be too tight and thereby miss some implied constraints. This guidance may have to be weakened.
- Allow PRESS, in tightly-controlled circumstances, to make external calls to a theorem prover (e.g., *Clam*, *Otter*) or a computer algebra system (e.g,. *Maple*) in order to further simplify or prune some constraints.

Experiments with the PRESS system

The following table shows a collection of examples. Some of the examples have been encountered in [7, 8, 9]. The first column shows the class of the implied constraints, the second the set of constraints, and the last is

[2] We can represent an attempted derivation as a tree. Each choice point creates a branch. The leaves are the formulae reached during deduction to which it could do no more and hence backtracked.

implied constraints.

Class	Constraints	Implied Constraints
C_1	$A+B+C=12$ $\wedge\ A \leq B$ $\wedge\ B \leq C$	$C \geq 4 \wedge A \leq 4$
C_2	$X \neq Y \wedge Y \neq Z \wedge X \neq Z$	$alldifferent([X,Y,Z])$
C_3	$A^2+B^2+C^2=12$ $\wedge\ A \geq 0$ $\wedge\ A \leq B \wedge B \leq C$	$(C \geq 2 \vee C \leq -2) \wedge (A \leq 2 \vee A \geq -2)$
C_3	$X*Y*Z=8 \wedge Y \leq Z$ $\wedge\ Z \leq 2$	$X \geq 2$
C_4	$A+B \neq C+D$	$X=A+B \wedge Y=C+D \wedge X \neq Y$
C_5	$S \cup Y = X$	$\mid S \mid + \mid T \mid \geq \mid X \mid$
C_6	$X=N-M \wedge Y=P-N$ $\wedge\ Z=P-M$	$Z=X+Y$
C_6	$X=A+B \wedge Y=C+D$ $\wedge\ A+C=1 \wedge B+D=3$	$X+Y=4$
C_6	$X=A+B \wedge Y=C+D$ $\wedge\ A+B \neq C+D$	$X \neq Y$

Our task now is to check how many of these implied constraints PRESS can produce. Here are the results:

Class	Constraints	PRESS results
C_1	$A+B+C=12 \wedge A \leq B \wedge B \leq C$	*yes*
C_2	$X \neq Y \wedge Y \neq Z \wedge X \neq Z$	*no*
C_3	$A^2+B^2+C^2=12 \wedge A \geq 0 \wedge A \leq B \wedge B \leq C$	*yes*
C_3	$X*Y*Z=8 \wedge Y \leq Z \wedge Z \leq 2$	*yes*
C_4	$A+B \neq C+D$	*no*
C_5	$S \cup Y = X$	*no*
C_6	$X=N-M \wedge Y=P-N \wedge Z=P-M$	*no*
C_6	$X=A+B \wedge Y=C+D \wedge A+C=1 \wedge B+D=3$	*no*
C_6	$X=A+B \wedge Y=C+D \wedge A+B \neq C+D$	*no*

In addition to being modular, the PRESS system has the following advantages:

- The PRESS system handles linear and nonlinear arithmetic constraints.
- The substitution mechanism of the PRESS system and its *isolation*, *collection*, and *attraction* rules allow it to handle very complex nonlinear constraints in a very nice way.
- PRESS can generate some implied constraints in classes C_1 and C_3.
- The PRESS system can be updated to handle constraints over other domains, such as the set domain, by adding the appropriate domain axioms as rewrite rules.

However, the PRESS system suffers from the following problems:

- PRESS is heavily biased to finding a *solution*. The guidance may be too tight and thereby miss some implied constraints in the classes C_1 and C_3.
- PRESS cannot handle classes C_2, C_4, C_5, and C_6.

2.3 The clp(q,r) Solver

Overview

The clp(q,r) solver [4, 5, 6] of SICStus Prolog [1] solves linear equations over rational or real valued variables, employs a decision algorithm for linear inequalities that derives implied constraints, handles linear disequations, and performs a lazy treatment for nonlinear equations, i.e., it collects them hoping that with the addition of further linear constraints they might get simple enough to solve.

Experiments with the clp(q,r) solver

The clp(q,r) solver, just like the PRESS system, tries to actually solve the provided set of constraints. However, what we are interested in is the implied constraints generated rather than the solutions. We carried out the same experiments as with the PRESS system and here are the results:

Class	Constraints	clp(q,r) results
C_1	$A+B+C=12 \wedge A \leq B \wedge B \leq C$	*no*
C_2	$X \neq Y \wedge Y \neq Z \wedge X \neq Z$	*no*
C_3	$A^2+B^2+C^2=12 \wedge A \geq 0 \wedge A \leq B \wedge B \leq C$	*no*
C_3	$X*Y*Z=8 \wedge Y \leq Z \wedge Z \leq 2$	*no*
C_4	$A+B \neq C+D$	*no*
C_5	$S \cup Y = X$	*no*
C_6	$X=N-M \wedge Y=P-N \wedge Z=P-M$	*yes*
C_6	$X=A+B \wedge Y=C+D \wedge A+C=1 \wedge B+D=3$	*yes*
C_6	$X=A+B \wedge Y=C+D \wedge A+B \neq C+D$	*yes*

As can be seen from these experiments, the clp(q,r) solver has a great potential to handle class C_6. However, it cannot handle classes C_1 to C_5.

3 Conclusion

In this technical report, we described experiments we performed with the PRESS solver for the generation of implied constraints. We found that with some modifications, in order to improve its ability to reason with inequalities, PRESS is capable of deriving useful implied constraints.

This technical report consists of the (unedited, apart from minor cosmetic, non-technical corrections) Sections 5.1, 5.2, and 5.3 of an unfinished technical report [3] written in August 2000, with the addition of an introduction and a conclusion.

References

[1] M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In: H. Glaser, P. Hartel, and H. Kuchen (eds), *Proc. of PLILP'97*, pp. 191–206. LNCS 1292. Springer-Verlag, 1997.

[2] C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints* 1(3):191–244, 1997.

[3] B. Hnich, J.D.C. Richardson, and P. Flener. Towards automatic generation and evaluation of implied constraints. Unfinished technical report, August 2000. Available as http://www.it.uu.se/research/group/astra/pub/IC0800.ps.gz.

[4] C. Holzbaur. A high-level approach to the realization of CLP languages. In: *Proc. of the JICSLP'92 Workshop on CLP Systems*, 1992.

[5] C. Holzbaur. A specialized, incremental solved form algorithm for systems of linear inequalities. Technical Report 94-07, Austrian Research Institute for Artificial Intelligence, Vienna, 1994.

[6] C. Holzbaur. OFAI clp(q,r) Manual, Edition 1.3.3. Technical Report 95-09, Austrian Research Institute for Artificial Intelligence, Vienna, 1995.

[7] L. Proll and B.M. Smith. ILP and constraint programming approaches to a template design problem. Technical Report 97.16, Dept of Computer Science, University of Leeds, UK, 1997.

[8] B.M. Smith, K. Stergiou, and T. Walsh. Modelling the Golomb ruler problem. In: *Proc. of the IJCAI'99 Workshop on Non-Binary Constraints*, 1999.

[9] K. Stergiou and T. Walsh. The difference all-different makes. In: *Proc. of IJCAI'99*, pp. 414–419. AAAI Press, 1999.

[10] L. Sterling, A. Bundy, L. Byrd, R. O'Keefe, and B. Silver. Solving symbolic equations with PRESS. *J. of Symbolic Computation* 7:71–84, 1989.