Generating Effective Constraint Programs: An Application of Automated Reasoning

Alan M. Frisch, Christopher Jefferson, Bernadette Martínez-Hernández, Ian Miguel
AI Group, Department Computer Science, University of York, York, England
{frisch, caj, berna, ianm}@cs.york.ac.uk

Introduction

Constraint programming has proven to be successful at solving a wide range of problems including important industrial problems. To solve a problem, one first "models" it by characterising its solutions by the constraints that they must satisfy. Then the highly-refined search methods of constraint programming can be used to search for a solution automatically. Unfortunately, a typical problem can be modelled in many ways, some of which may be much easier to solve than others. Formulating an effective model requires considerable expertise and is an impediment to the take-up of constraint programming.

We are addressing this so-called "modelling bottleneck" by developing an intelligent compiler that can automatically refine a high-level problem specification into a model that can be solved effectively by existing constraint solvers. This paper presents our work as an exercise in automated reasoning (AR) and from this viewpoint focuses on the issue of designing an appropriate architecture for this form of reasoning.

A model, abstract or concrete, can be read declaratively as an existentially quantified conjunction of constraints; the goal is to find values for the existentially-quantified variables that satisfy the conjunction. Our intelligent compiler—of which we have a running prototype [1]– is based on a set of compositional refinement rules that capture some of the expertise of human modellers. Thus, refinement rules map an abstract model to a logically equivalent concrete one. It is not hard to see that our work is applied AR and, interestingly, we are encountering many issues that arise elsewhere in AR.

Effective modelling involves more than just refinement, and again the issues are familiar in the field of AR. For example, models can be made more efficient by judiciously adding constraints that are implied by others in the model—deduction! Models often contain many symmetries, which result in symmetries in the search space. Consequently, expert modellers often add symmetry-breaking constraints to a model, which can drastically reduce the size of the search space. Again, this is common in theorem proving—for example, "consider x and y and without loss of generality let x be the smaller of the two."

The Architecture of Reasoning

To this point our implemented systems work by applying rule in a forward chaining manner. Our initial system, CGRASS [4], takes a set of constraints and augments it with implied constraints. Each rule identifies a pattern in the set of constraints and either adds new constraints or replaces existing constraints. Each rule is coupled with and is governed by heuristics that identify when the transformation it performs is likely to improve the model. CGRASS applies its rules in a forward-chaining manner until no other rules apply.

The rules in our refinement system are compositional; the refinement of an expression is composed of the refinements of its parts. Structurally these rules differ greatly from those in CGRASS, yet their application can also be seen as forward chaining. At each point, from among all the rules that are applicable, one is chosen and applied. In most situations multiple rules are applicable because there are many possible refinements. It can be very difficult to identify which of the alternative ultimately lead to the best model; the system does not yet have heuristics for choosing good rules from among the alternatives.

As we begin to study the problem of automating rule selection, we are noticing cases where a rule is useful because it enables subsequent transformations that are highly effective. Each of the following three subsections gives example of this. The first two subsections give examples where a symmetry-breaking constraint enables a useful transformation. The third example shows how one refinement enables a powerful transformation that would not be available using an alternative refinement.

These examples suggest that a simple forward-chaining architecture is inadequate for the reasoning that is involved in generating effective models.

Symmetry-breaking to Select a Disjunct

If a constraint program contains (or entails) a disjunction, say $C_1 \vee \cdots \vee C_n$, we can, in some cases, introduce a symmetry-breaking constraint so that the resulting constraint program entails a particular disjunct, C_i. We can then replace the weak disjunction with the strong constraint, C_i.

Consider the following example. The 3-fractions puzzle (see problem 41 at www.csplib.org for the general form of the puzzle) is to find nine distinct non-zero digits, A to I, that satisfy:

$$\frac{A}{BC} + \frac{D}{EF} + \frac{G}{HI} = 1 \tag{1}$$

where BC is short for $10 \times B + C$, *etc.*

The following disjunction can be derived:

$$(\frac{A}{BC} \leq \frac{1}{3}) \vee (\frac{D}{EF} \leq \frac{1}{3}) \vee (\frac{G}{HI} \leq \frac{1}{3}) \tag{2}$$

Otherwise, the sum of the fractions would exceed 1.

Constraint (2) can be strengthened by breaking symmetry. The symmetry in this problem comes from the associativity and commutativity of the sum operator, and the fact that the variables have identical domains. Any (non-)solution can be transformed into a (non-)solution by exchanging the values assigned to one fraction for another. This symmetry can be broken by ordering the fractions:

$$\frac{A}{BC} \leq \frac{D}{EF} \leq \frac{G}{HI} \qquad (3)$$

Given (2) and (3), it is possible to derive:

$$\frac{A}{BC} \leq \frac{1}{3} \qquad (4)$$

Note that (3) is just one of many symmetry-breaking schemes. However, if the modeller is aware of implied constraints such as (2), this can guide the choice of symmetry-breaking constraints.

Symmetry-breaking to Strengthen Inequalities

If a constraint program has symmetric terms that cannot be "equivalent" under some ordering in a solution, then a strict ordering can be imposed among the terms. The justification is that the symmetry can be broken by imposing a non-strict ordering on the terms and, since the terms cannot be equivalent, the non-strict ordering can be strengthened to a strict ordering.

As an example, notice that in the 3-fractions puzzle an alternative symmetry-breaking constraint is

$$A \leq D \leq G \qquad (5)$$

Since the problem requires that each variable is assigned a different value, this symmetry-breaking constraint can be replaced with a stronger one:

$$A < D < G \qquad (6)$$

Again, if the modeller is aware of this simplification, the symmetry-breaking constraints can be chosen to suit.

The current practice among modelling experts is to chose among possible symmetry-breaking constraints by choosing the one that provides the best tradeoff between the amount of symmetry broken and the overhead of it entails. Using this fractions problem as an example, we have shown [3] that the choice also needs to consider the implied constraints that the symmetry-breaking constraint enables. Considered in isolation, a model with symmetry-breaking constraint (3) is less effective than one with (5). Yet if we add the implied constraints (4) and (6) to the two models, respectively, the situation reverses: the first model becomes the more effective one.

Simplifying If-then Constraints

When refining an abstract specification into a constraint program, there are often several ways of representing an abstract object. The Golomb Ruler problem (problem 6 in CSPLib) is to find a set T of n integers such that:

$$\forall \{i,j\}, \{k,l\} \subseteq T . \{i,j\} \neq \{k,l\} \rightarrow |i-j| \neq |k-l| \qquad (7)$$

where $\{i,j\} \subseteq T$ means that $\{i,j\}$ is a two-element subset of T. This constraint is abstract since the decision is in terms of a set rather than atomic variables. Hence, it must be refined into a constraint program.

Since the size of T is known, one possible refinement is to a one-dimensional matrix, $T[\,]$ with one element per element of T, coupled with an all-different constraint on the elements of the matrix to ensure that the set is represented properly:

$$\text{AllDifferent}(T[\,]) \qquad (8)$$

$$\forall \{a,b\}, \{c,d\} \subseteq I . (\{T[a], T[b]\} \neq \{T[c], T[d]\}) \rightarrow (|T[a] - T[b]| \neq |T[c] - T[d]|) \qquad (9)$$

where I is the set of indices into $T[\,]$.

Since the elements of $T[\,]$ are all-different, tests for equality and disequality, such as $T[a] = T[b]$ and $T[a] \neq T[b]$, can be simplified to $a = b$ and $a \neq b$ respectively. This reasoning lifts to sets in a straightforward manner:

$$\forall \{a,b\}, \{c,d\} \subseteq I . (\{a,b\} \neq \{c,d\}) \rightarrow (|T[a] - T[b]| \neq |T[c] - T[d]|) \qquad (10)$$

This gives a much simplified set of constraints: since $\{a,b,c,d\}$ is known statically, only the consequent part of (10) need be posted as a constraint for each appropriate set of values, $\{a,b,c,d\}$.

Consider an equivalent alternative specification:

$$\forall i,j,k,l . (\{i,j\} \subseteq T \wedge \{k,l\} \subseteq T \wedge \{i,j\} \neq \{k,l\}) \rightarrow (|i-j| \neq |k-l|) \qquad (11)$$

If T is again represented as a one-dimensional matrix, $T[\,]$, refinement of the first conjunct in the antecedent gives:

$$\begin{aligned} i \neq j \quad & \wedge \\ i = T[1] \vee i = T[2] \vee \cdots \vee i = T[n] \quad & \wedge \\ j = T[1] \vee j = T[2] \vee \cdots \vee j = T[n] \quad & \end{aligned} \qquad (12)$$

Similarly for the second conjunct. The difference from the first refinement is that i, j, k and l are not drawn from T itself, so cannot be guaranteed to be in the refined set. The result is much less attractive as a constraint, and requires much more effort to simplify. In a situation like (11), it is therefore useful to move the first two conjuncts into the restriction to produce (7).

Conclusion

Formulating an effective model of a constraint program is a challenging task for AR. We have made some good progress towards this goal by building systems that reason by forward chaining. The examples presented here suggest that we may need to employ a more-complex reasoning architecture that incorporates some goal-based reasoning. We wonder whether an architecture based on proof-planning [2] would be appropriate.

Acknowledgements We thank Toby Walsh, who played a central role in the development of CGRASS, and Adam Bakewell, who played a central role in the development of the refinement system.

References

[1] A. Bakewell, A.M. Frisch, I. Miguel. Towards automatic modelling of constraint satisfaction problems: A system based on compositional refinement. *Proc. 2nd Int. Wshop on Modelling and Reformulating CSPs*, 2-17, 2003.

[2] A. Bundy. A Science of Reasoning. *Computational Logic: Essays in Honor of Alan Robinson*, 178-198, 1991.

[3] A.M. Frisch, C. Jefferson, I. Miguel. Symmetry Breaking as a Prelude to Implied Constraints. Under review, 2004.

[4] A.M. Frisch, I. Miguel, T. Walsh. CGRASS: A System for Transforming Constraint Satisfaction Problems. *Proc. ERCIM/Colognet Int. Workshop on Constraint Solving and CLP* (LNAI 2627), 15-30, 2002.