

# Refining and Reformulating an Abstract Specification of the Golomb Ruler Problem

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

## 1 Introduction

Constraint programming has been very successful in tackling a wide variety of combinatorial problems in industry and academia. However, to apply constraint programming tools to a domain, the problem must be *modelled* as a constraint program. Typically, a problem can be modelled in a variety of ways, some more effective than others. Hence, constructing an effective model is difficult, requiring a great deal of expertise. Automating this process is therefore a significant challenge. The focus of this report is on demonstrating the utility of the *refinement* approach to this problem: the transformation of an abstract specification of a problem into a set of alternative constraint programs. This provides the foundation for future work in which the set of alternatives will be ranked in order to provide the best model or models for a particular problem.

## 2 Background

The finite domain *constraint satisfaction problem* (CSP, [2]) consists of a triple  $\langle X, D, C \rangle$ , where  $X$  is a set of variables,  $D$  is a set of domains, and  $C$  is a set of constraints. Each  $x_i \in X$  is associated with a finite domain  $D_i \in D$  of potential values. A variable is *assigned* a value from its domain. A constraint  $c \in C$ , constraining variables  $x_i, \dots, x_j$ , specifies a subset of the Cartesian product  $D_i \times \dots \times D_j$  indicating mutually compatible variable assignments. A solution is an assignment to all elements of  $X$  that satisfies all the constraints.

The abstract specification language and refinement mechanism employed here was introduced in Bakewell *et al.* [1]. To recap, a problem specification is a triple consisting of a list of *declarations*, a set of *decision variables*, and a set of *constraints*. *Refinement rules* transform specifications into constraint programs. Each rule specifies, by means of a refinement operator  $\rho$ , the refinement of an expression in the specification language to a set of alternative corresponding expressions in a language similar to a constraint program. The rules are *compositional*: specifications are refined by refining and combining each of their constituent parts.

As an example, consider the refinement of a binary relation  $R: \wp(D_1 \times D_2)$ . The type of  $R$  is the powerset of the Cartesian product of its argument sets,  $D_1$  and  $D_2$ , written  $\wp(D_1 \times D_2)$ . The following rule refines  $R$  to a Boolean matrix,  $R_{M01}$ :

RELATION:  $\rho(R: \wp(D_1 \times D_2)) \stackrel{\text{ref}}{\rightarrow} \{\lambda d_1, d_2. R_{M01}[d_1, d_2]: D_1 \times D_2 \rightarrow 01\}$

This is a considerable simplification of the actual rule. First, in practice the rules are able to handle relations of arbitrary arity. Second, for the rules to be compositional,  $D_1$  and  $D_2$  must also be refined (potentially in a variety of ways, leading to the set returned by RELATION containing a number of elements, each expressing an alternative refinement of  $R$ ). Finally, other clauses of the RELATION rule support alternative refinements of a relation, such as a one-dimensional matrix indexed by  $D_1$  and containing elements of type  $D_2$ . This production-rule style is a development of that given previously [1].

### 3 The Golomb Ruler Problem

The Golomb ruler problem (problem 006 at [www.csplib.org](http://www.csplib.org)) is a challenging combinatorial problem with many practical applications, such as radio astronomy and X-ray crystallography [3]. It is particularly interesting from a modelling point of view because it illustrates the utility refining an abstract specification to produce a model for all instances of a problem class, versus the reformulation of a basic model of an individual instance. The latter approach was explored previously [5], where it was found that the overhead of reformulating each instance of a class of problems prior to solving was considerable.

The problem is succinctly specified in natural language:

Given  $n$ , put  $n$  ticks on a ruler of size  $m$  such that all inter-tick distances are unique.  
Minimise  $m$ .

There are several ways in which this problem can be translated into the abstract specification language. A straightforward possibility is:

```
given  $n:nat$ 
letting  $R$  be rangetype  $0..2^n$ 
find  $T:\wp_n(R)$ 
such that
(1)  $\forall \{i,j\} \subseteq T. \forall \{k,l\} \subseteq T. \{i,j\} \neq \{k,l\} \rightarrow |i-j| \neq |k-l|$ 
(2)  $Minimise(max(T))$ 
```

The remainder of this report demonstrates how this specification can be refined effectively into a constraint program.

### 4 Refinement and Reformulation of the Golomb Ruler Problem

This section focuses primarily on the ticks constraint (1), which is the most complex to refine. *Reformulation* is distinguished from refinement as being a transformation from one expression or set of expressions into another at the same level of abstraction. Reformulation rules are used throughout the refinement process to improve efficiency and eliminate redundancy, as will be shown.

#### 4.1 Refinement of a Universally Quantified Expression

Constraint (1) has the general form  $\forall s \subseteq S:\wp(\tau).X:bool$ . Hence, a refinement rule is necessary to support universal quantification using subset as a binding expression. The rule has three cases, depending on the way in which the superset  $S$  is refined (note that sets are treated as unary relations, and refined by the RELATION rule). The three clauses operate basically in the same way: the superset  $S:\tau$  is first refined to  $S':\tau'$ , then, for each element  $a_i:\tau$  of the set of quantified variables, a new element  $a'_i:\tau'$  is created (the third clause is a little more complex, but is essentially the same). To construct the final refined expression,  $X$  is then refined in the *context* (denoted by the subscript of  $\rho$ ) where each free occurrence of  $a_i$  in  $X$  is bound to  $a'_i$ . Note that sized sets (a set of size  $k$  is denoted by the type  $\wp_k$ ) are a sub-type of the set type. All refinements that can be performed on sets can be performed on sized sets also. Hence, for sized sets, only the refinement exclusive to the sized set type is described below.

The numbering scheme in the comprehension denotes the order in which the expressions are to be evaluated. This is a device employed for clarity, equivalent to nesting of set comprehensions. As described previously [1], by default refinement returns a local model consisting of type definitions, annotations and constraints. For brevity, this local model is only displayed if accessed by the current rule, as opposed to being returned intact. Finally, note that the type of a matrix indexed by elements of type  $\tau$  and containing elements of type  $\tau'$  is denoted  $[\tau]_{\tau'}$ .

- $S$  remains a set.

$$\begin{array}{l} \text{FORALLSUBSET: } \rho_c(\forall\{a_1, \dots, a_n\} \subseteq S:\wp(\tau).X:\text{bool}) \stackrel{\text{ref}}{\dashv} \\ \quad \{\forall\{a'_1, \dots, a'_n\}:\wp(\tau').((a'_1:\tau' \in S':\wp(\tau')) \wedge \dots \wedge (a'_n:\tau' \in S':\wp(\tau')))) \rightarrow (X':\text{bool}) \\ \quad | \\ \quad 1. S':\wp(\tau') \in \rho(S:\wp(\tau)) \\ \quad 2. X':\text{bool} \in \rho_{c[a_1 \mapsto a'_1:\tau', \dots, a_n \mapsto a'_n:\tau']}(X) \\ \quad \} \end{array}$$

- $S$  is refined to a Boolean/0-1 Array.

$$\begin{array}{l} \text{FORALLSUBSET: } \rho_c(\forall\{a_1, \dots, a_n\} \subseteq S:\wp(\tau).X:\text{bool}) \stackrel{\text{ref}}{\dashv} \\ \quad \{\forall\{a'_1, \dots, a'_n\}:\wp(\tau').(S'[a'_1] \wedge \dots \wedge S'[a'_n]) \rightarrow (X':\text{bool}) \\ \quad | \\ \quad 1. S':[\tau']_{\text{bool}} \in \rho(S:\wp(\tau)) \\ \quad 2. X':\text{bool} \in \rho_{c[a_1 \mapsto a'_1:\tau', \dots, a_n \mapsto a'_n:\tau']}(X) \\ \quad \} \end{array}$$

- $S_k$  is refined to a one-dimensional matrix indexed by a type of size  $k$ .

$$\begin{array}{l} \text{FORALLSUBSET: } \rho_c(\forall\{a_1, \dots, a_n\} \subseteq S:\wp_k(\tau).X:\text{bool}) \stackrel{\text{ref}}{\dashv} \\ \quad \{\forall\{i_1, \dots, i_n\}:\wp_n(I).(X' : \text{bool}) \\ \quad | \\ \quad 1. (S':[I]_{\tau'} \text{ letting } I \text{ be type of size } k) \in \rho(S:\wp(\tau)), \\ \quad 2. X':\text{bool} \in \rho_{c[a_1 \mapsto S'[i_1:I], \dots, a_n \mapsto S'[i_n:I]}(X) \\ \quad \} \end{array}$$

## 4.2 Applying the FORALLSUBSET Rule

It is useful at this point to apply the FORALLSUBSET rule to constraint (1) of the abstract specification of the Golomb ruler given in Section 3. For brevity, only on refinement is shown for each of the rules given in the previous sub-section. Others are possible, combining different representations of the set of ticks into a single model, but are not discussed at present.

First consider a simple refinement where  $T$  (the set of ticks) remains a set:

$$\begin{array}{l} \text{given } n:\text{nat} \\ \text{letting } R \text{ be rangetype } 0..2^n:\text{nat} \\ \text{find } T:\wp(R) \\ \text{such that} \\ \quad \forall \{i, j\}:\wp(R). \forall \{k, l\}:\wp(R). ((i:R \in T) \wedge (j:R \in T)) \wedge ((k:R \in T) \wedge (l:R \in T)) \rightarrow \\ \quad \quad (\{i, j\} \neq \{k, l\} \rightarrow |i - j| \neq |k - l|) \end{array}$$

The difference is that the quantified variables have a primitive type  $R$  rather than being elements of  $T$ , a decision variable, in the original specification. This is important because the refined form can be transformed straightforwardly into nested for loops.

The refinement to a 0/1 array to represent the ticks proceeds similarly. An additional sum constraint is necessary to ensure that the 0/1 representation of the set contains the right number of elements. This constraint is generated by the RELATION rule for sized relations.

$$\begin{array}{l} \text{given } n:\text{nat} \\ \text{letting } R \text{ be rangetype } 0..2^n:\text{nat} \\ \text{find } T:[R]_{\text{bool}} \\ \text{such that} \\ \quad \sum_{r:R} T[r] = n \\ \quad \forall \{i, j\}:\wp(R). \forall \{k, l\}:\wp(R). (T[i:R] \wedge T[j:R]) \wedge (T[k:R] \wedge T[l:R]) \rightarrow \\ \quad \quad (\{i, j\} \neq \{k, l\} \rightarrow |i - j| \neq |k - l|) \end{array}$$

Finally, consider the refinement to a one-dimensional matrix indexed by  $I$ , a type with  $n$  elements, one for each element of the original size  $n$  set. The AllDifferent constraint is necessary to make sure that the matrix properly represents a set, and is again produced by the RELATION rule applied to a sized set.

```

given  $n:nat$ 
letting  $R$  be rangetype  $0..2^n:nat$ ,  $I$  be type of size  $n$ 
find  $T:[I]_R$ 
symIndex ( $T:[I]_R$ , 1)
such that
  AllDifferent( $\{T[i]|i:I\}$ )
   $\forall\{a, b\}:\wp_2(I).\forall\{c, d\}:\wp_2(I).((\{T[a], T[b]\} \neq \{T[c], T[d]\}) \rightarrow (|T[a]-T[b]| \neq |T[c]-T[d]|))$ 

```

This refinement has introduced a new symmetry into the problem. Indices of the one-dimensional matrix are distinguished; one index refers to the first tick, another the second, and so on. The abstract set representation had, of course, no such index labels. This gives rise to a symmetry whereby, in any (non-)solution, the assignments to elements of the ticks matrix may be permuted to obtain another (non-)solution. Since, however, this symmetry is introduced by refinement itself, it is a simple matter to annotate the model with this fact using the ‘symIndex’ statement. In this case, symIndex ( $T:[I]_R$ , 1) records the fact that the first index of  $T:[I]_R$  has symmetry. The symmetry can then be broken by any of a variety of static or dynamic methods. Compare this simple process with the expensive symmetry-detection approach adopted by the CGRASS system on instances of the Golomb Ruler problem [5].

The remainder of this report focuses on this last refinement, since it is closest to the model usually formulated by experts. As will be discussed, it is possible to combine reformulation and further refinement to improve this part of the model significantly.

### 4.3 Reformulation Leading to Further Refinement

Reformulation rules, similar to those employed by the CGRASS system [5], are applied to each element of the set of expressions returned by an application of the  $\rho$  operator. The difference here is that the reformulation rules are able to work at a higher level of abstraction.

Consider the constraints governing the inter-tick distances in the final refinement above:

```

AllDifferent( $\{T[i]|i:I\}$ )
 $\forall\{a, b\}:\wp_2(I).\forall\{c, d\}:\wp_2(I).((\{T[a], T[b]\} \neq \{T[c], T[d]\}) \rightarrow (|T[a]-T[b]| \neq |T[c]-T[d]|))$ 

```

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, giving:

```

AllDifferent( $\{T[i]|i:I\}$ )
 $\forall\{a, b\}:\wp_2(I).\forall\{c, d\}:\wp_2(I).((\{a, b\} \neq \{c, d\}) \rightarrow (|T[a] - T[b]| \neq |T[c] - T[d]|))$ 

```

Having made this simplification, a further powerful reformulation is possible. Briefly, the GENALLDIFF reformulation rule has the following form:

```

In:     $\forall X:\tau.\forall Y:\tau.((X \neq Y) \rightarrow (f(X) \neq f(Y)))$ 
Out:   $\rho(\text{AllDifferent}(\{f(X)|X:\tau\}))$ 

```

Applying this rule to the example gives:

```

 $\rho(\text{AllDifferent}(\{|T[a] - T[b]| \mid \{a, b\}:\wp_2(I)\}))$ 

```

Replacing the clique of not-equals constraints with an all-different constraint results in a model with far greater pruning power [6]. This relatively simple transformation may be contrasted with the corresponding instance-level reformulation. At the instance level, the clique structure must

first be detected. Maximal clique finding is an NP-complete problem, hence an instance-based reformulation system must rely on an approximation algorithm [5]. Therefore, obtaining the optimal pruning model is not guaranteed and still relatively expensive (especially as the clique-finding process must be repeated at each instance) compared with the abstract reformulation just discussed.

Finally, it is necessary to refine the output of the GENALLDIFF rule, since constraint programs support all-different applied to CSP variables only. The ALLDIFFERENT refinement rule is partially specified as follows:

$$\text{ALLDIFFERENT: } \rho(\text{AllDifferent}(X:\wp_k(\tau))) \xrightarrow{\text{ref}} \left\{ \begin{array}{l} \text{AllDifferent}(X') \\ | \\ 1.(X':[I]_\tau \text{ letting } I \text{ be type of size } k) = \text{VARINTRODUCE}(X) \\ \} \end{array} \right.$$

The VARINTRODUCE reformulation rule returns an array of variables, each of which is constrained to be equal to an element of the input set.

In the example, the size of the input set to the all-different constraint is simply  $\binom{n}{2}$ . Applying the ALLDIFFERENT refinement rule gives the final model:

```

given  $n:nat$ 
letting  $R$  be rangetype  $0..2^n:nat$ ,  $I$  be type of size  $n$ ,  $I'$  be type of size  $\binom{n}{2}$ ,
       $f$  be an arbitrary bijection from  $I'$  to  $\wp_2(I)$ 
find  $T:[I]_R$ ,  $D:[I']_R$ 
symIndex ( $T:[I]_R$ , 1)
such that
  AllDifferent( $\{T[i]|i:I\}$ )
  AllDifferent( $\{D[i']|i:I'\}$ )
   $\forall i':I'.\{a,b\} = f(i') \rightarrow D[i'] = |T[a] - T[b]|$ 

```

The bijection  $f$  is introduced by the VARINTRODUCE reformulation rule above. This function will vary according to the set comprehension input to the reformulation rule. It is straightforward to reformulate  $f$  into nested for loops suitable for use with most constraint programming languages.

This model corresponds closely to that employed by human modellers [7]. Furthermore, it specifies an entire *class* of problems, despite involving less effort to create from a basic abstract specification than was previously used to reformulate just one instance of a basic model [5].

## 5 Conclusion

This report has concerned the refinement and reformulation of the Golomb Ruler problem. In particular, it has demonstrated the utility of the refinement approach versus the previously adopted method of refining individual instances. It has also demonstrated the efficacy of reformulating the problem at a higher level of abstraction than the instance level, as first suggested in [4]. Refinement and reformulation were interleaved, leading to an effective derivation of a good model. It is expected that this pattern will be repeated as this approach is applied to other challenging modelling problems, and the set of models refined from this and similar abstract specifications of the Golomb Ruler problem are explored.

## References

- [1] A. Bakewell, A.M. Frisch, I. Miguel. Towards automatic modelling of constraint satisfaction problems: a system based on compositional refinement. *Proceedings of the 2nd International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pp. 2–17, 2003.

- [2] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
- [3] A.K. Dewdney. Computer Recreations. In *Scientific American*, pp. 16–20, December 1985.
- [4] A.M. Frisch, B. Hnich, I. Miguel, B.M. Smith, T. Walsh. Towards Model Reformulation at Multiple Levels of Abstraction. *Proceedings of the International Workshop on Reformulating Constraint Satisfaction Problems*, pp. 42–56, 2002.
- [5] A.M. Frisch, I. Miguel, T. Walsh. CGRASS: A System for Transforming Constraint Satisfaction Problems. In *Proceedings of the ERCIM/Colognet International Workshop on Constraint Solving and Constraint Logic Programming (LNAI 2627)*, pp. 15–30, 2003.
- [6] J.C. Régim. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pp. 362–367, 1994.
- [7] B.M. Smith, K. Stergiou, T. Walsh. Using Auxiliary Variables and Implied Constraints to Model Non-binary Problems. In *Proceedings of the 16th National Conference on Artificial Intelligence*, pp. 182–187, 2000.