The Automatic Generation of Redundant Representations and Channelling Constraints

Bernadette Martínez-Hernández and Alan M. Frisch

Department of Computer Science
University of York
United Kingdom
{berna, frisch}@cs.york.ac.uk

Abstract. Constraint modelling is the process of encoding a given problem to be solved by constraint satisfaction technology. Automatic modelling systems aim to reduce the number of decisions human modellers must take. To do so, these systems implement common modelling guidelines and techniques. In this paper we focus on the automatic addition of redundant information and most importantly the corresponding channelling constraints to synchronise it. We discuss and formalise a systematic method of generation of both elements; redundancy and channelling constraints. We provide in this paper a new insight on this formalisation that aims to clarify and increase previous work on the subject [1].

1 Introduction

Constraint modelling is the process of encoding a problem into finite-domain decision variables and a set of constraints posed over these variables. A model used to solve a constraint satisfaction problem is normally not unique. In many cases, the way we model a problem determines how fast we find a solution or whether we find one at all. Modellers propose and test several alternative models to finally select the most efficient ones to solve the general problem. During the modelling process, alternative models are created by means of formulating the problem with different sorts of variables and constraints.

Efficient constraint modelling is a hard task often learnt by novice constraint users from modelling examples. In order to reduce the time spent on modelling, many automatic modelling systems have arisen. The construction one of these automatic systems requires a good understanding of the modelling process as well as each commonly used modelling technique. One of these frequently used modelling techniques is the addition of *redundant representations* and *channelling constraints*. These constraints were defined by Cheng [2] and, although widely used, little has been done to study them thoroughly from a point of view that systematically associates added redundant information to the needed channelling constraint (channel). Previously, we introduced an algorithm of generation of channels [1]. This algorithm, even though correct, lacked of a clear association to the redundant elements added to the model. We seek to clarify in

this paper the relation between redundant representations and channels in order to describe the necessary requirements to add channelling constraints.

In the remainder of this introduction we explain what channelling constraints are. At the end of this section we introduce the main issues of channelling addressed in this paper.

Let us now use the well known *n-queens* problem as an example of modelling. To solve this problem we need to find a configuration of n queens on an $n \times n$ board such that no queen is under attack. Nadel [3] has already discusses several alternative approaches that can be used to solve this problem. We use two of the models he describes in his paper, and afterwards we detail a third *combined* model.

The first approach we use in this paper uses an array R of n variables to encode the n rows of the chess board. Each variable of the array has an integer domain $1..n$ too, that is, each element of the domain corresponds to one of the n columns of the board. Hence, $R[i] = j$ when a queen is placed on row i and column j.

A second approach swaps the column-values for the row-variables obtaining the array of n variables C. The variables of the array C have an integer domain $1..n$. For this model, we say $C[i] = j$ when a queen is placed on the column i and row j.

Solving simultaneously both approaches[1] but disregarding any attempt to synchronise their assignments to variables, may only cause overhead due to the lack of connection between C and R to propagate the values pruned in each group of variables. Thus we introduce channelling constraints or channels to maintain the consistency between C and R. The following channel ensures simultaneous assignments of values to variables of each model stand for the same 'abstract' value in the chess board, that is, an assignment of a queen to the position row i and column j should be occur in both models in the combination at the same time.

$$\forall i{:}1..n \;.\; \forall j{:}1..k \;.\; R[i] = j \Leftrightarrow C[j] = i \tag{1}$$

This channel connects matrices of one dimension, but this is not the only kind of channels that occur. Variable structures of several dimensions and/or integer sets may require more intricate channels to establish the propagation link. For instance, consider the Sonet problem [4] where a network configuration needs to be found. The Sonet network has a certain number of rings (*nrings*) and nodes (*nnodes*) and each of its rings can take nodes up to a defined limit (*capacity*). For simplicity, in this example we discard the constraints and optimisation function of the complete Sonet problem. As well as with the n-queens problem, we show two alternative approaches to model the network and their channelling constraints.

[1] There are `alldifferent` constraints imposed on both C and R, but we do not include in this example for simplicity.

X	$= rings$
$D_X(rings)$	$= multisets\ (of\ size\ nrings)\ of\ sets\ (of\ maxsize\ capacity)\ of\ 1..nnodes$
C	$= \emptyset$

Fig. 1. CSP-instance 1 of the rings of the Sonet problem.

The first approach we describe encodes the set of rings as a Boolean 2-dimensional matrix $rings_1$ indexed by $1..nrings$ and $1..nnodes$; where $rings_1[i, j] = True$ if node j is in ring i.[2]

The second modelling approach uses two 2-dimensional matrices, $rings_2$ and $switch$ of integer and Boolean variables respectively. Both are indexed by $1..nrings$ and $1..capacity$; and the domain of each variable in $rings_2$ is $1..nnodes$. Node j is in ring i if there is some k such that $rings_2[i, k] = j$ and $switch[i, k] = True$.

Simultaneous solving of both approaches require the addition of the following channel to maintain the consistency between assignments:

$$\forall i \in 1..nrings\ .\ \exists l \in 1..capacity\ .\ (rings_1[i, rings_2[i, l]] \wedge switch[i, l]) \quad (2)$$

The channelling constraint (1) ensures node k is on ring i for $rings_1$ ($rings_1[i, k] = True$) if and only if it is also on ring i according to the model with ($rings_2$, $switch$) (that is there is a position l such that $rings_2[i, l] = k$ and $switch[i, l]$ has truth value of $True$). Note the channel also imposes the same order on the solutions on both linked approaches by using the same index i in the channel.

Each of the alternatives was constructed by taking modelling decisions about how to encode the Sonet network, even when we did not mention them precisely. These decisions provide a guideline that describes the connection between the variables used in the encoding and the problem to encode. Then, to fully describe the formal steps involved in the correct generation of channelling constraints we answer several questions: what is a correct representation? in Section 2; what are channels? in Section 3; how do we generate alternative redundant representations? in Section 4; given two redundant representations how do we automatically generate the channelling constraints to link them, and how can the process of generation of representations be used to build a correct channels? in Section 5; how do we generate an efficient implementation of a channel? in Section 6. Conclusions and future work are presented in the last section.

2 Representations

Any constraint model of a problem is formulated to *represent* the problem. In this paper we focus on CSP instances, that is, triples of variables, domains and constraints (X, D_X, C) representing only *one* instance of a problem. For this

[2] Notice constraints must be added to guarantee the rings do not surpass their capacity. We discard these constraints in this section.

X_1	$= rings_1$
$D_{X_1}(rings_1)$	$=$ *2-D matrix of Boolean indexed by 1..nrings and 1..nnodes*
C_1	$= \forall\ i \in 1..nrings\ .\ ((\sum j \in 1..nnodes\ .\ rings_1[i,j]) = capacity)$

Fig. 2. CSP-instance 2 of the rings of the Sonet problem.

X_2	$= rings_2$, *switch*
$D_{X_2}(rings_2)$	$=$ *2-D matrix of 1..nnodes indexed by 1..nrings and 1..capacity*
$D_{X_2}(switch)$	$=$ *2-D matrix of Boolean indexed by 1..nrings and 1..capacity*
C_2	$= \forall\ i \in 1..nrings\ .$ `alldifferent`$(rings_2[i])$

Fig. 3. CSP instance 3 of the rings of the Sonet problem.

reason, to illustrate the notion of representation we assume all the parameters (capacity and number of nodes and rings) of the Sonet miniature problem of Section 1 are instantiated to some values. We describe in this section several CSP instances formulating this assumed instance of the problem.

The CSP instance of Figure 1 consists of a variable whose domain is composed of multisets (groups of elements with (possibly) multiple occurrences). The elements of the multisets in the domain of this variable are sets of integers. This type of variable (multiset of sets of integer) cannot be implemented directly in any current solver, however we disregard this fact in this paper when formulating CSP instances. The domain of the *rings* variable comprises all the possible configurations of the network of rings by considering each set as a ring of nodes.

CSP instance 2, shown in Figure 2, also characterises the network of rings. Each ring of the network is represented by a 2-dimensional matrix of Booleans in a fashion already described in Section 1.

Both CSP instances represent the problem instance. Moreover, each solution of the CSP instance 2 can be mapped into a solution of the CSP instance 1. Intuitively, this mapping describes how CSP instance 2 *represents* CSP instance 1. Following this idea of having a mapping between solutions, we introduce our definition of representation for CSP instances as follows.

Definition 1 *R' **represents** R via ψ, if $R' = (X', D_{X'}, C')$ and $R = (X, D_X, C)$ are CSP instances and ψ is a partial function from the total assignments of X' into the total assignments of X such that:*

- *For each total assignment w' of the variables in X', w' is a solution of C' **if and only if** $\psi(w')$ is defined and it is a solution of C,*
- *For each solution w of C, there is at least one solution w' of C' such that $\psi(w') = w$.*

*We say that R' **represents** or **is a representation** of R if for some ψ R' represents R via ψ.*

In our example, CSP instance 2 represents CSP instance 1 via the function ψ from assignments of $rings_1$ to assignments of $rings$. The function ψ transforms each of the rows of the Boolean matrix into sets composed of the indexes of cells assigned to *True*.

Our definition of representation is strongly influenced by definition of *variable representation* introduced by Jefferson and Frisch [5]. Their variable representation has attached the surjective mapping used to preserve the solutions but discards any constraint imposed on any of the variables.

3 Alternative Representations and Channels

Constraints provide a rich language that allows many different ways to solve a problem. Each alternative CSP instances used to solve a problem instance are some how related by the problem instance they represent. We show in this section that a connection between two CSP instances can be established if they both represent (separately) a third CSP instance. To clarify this let us now introduce the CSP instance of Figure 3 containing matrices $rings_2$ and $switch$ of integer and Boolean variables respectively; where node j is in ring i if for some k, $rings_2[i,k] = j$ and $switch[i,k] = True$. This instance models all the network configurations of our Sonet problem instance too. Moreover, CSP instance 3 also represents CSP instance 1 via a function that maps the row values of the elements activated by $switch$ into sets of a multisets. Since CSP instance 2 and CSP instance 3 represent CSP instance 1 we say CSP instance 2 and 3 are *redundant* with respect to CSP instance 1.

Our definition of redundancy with respect to a third CSP instance ensures both instances can be linked to each other via a third instance, but avoiding describing the relation connecting solutions of each redundant instance.

Two CSP instances are *equivalent* when they both represent each other. Notice redundancy does not necessarily entail equivalence. Limiting our interest to equivalent CSP instances is very restrictive, the spectrum of equivalent representations is very reduced, lacking many useful representations. For example, CSP 2 and 3 are not equivalent since CSP 2 cannot represent CSP 3. Furthermore, redundant representations are sufficient to describe many modelling transformations and thus detail systematic modelling. More importantly, (alternative) redundant representations does not need to be equivalent.

The definition of redundancy of Law and Lee in their Model Induction [6] and Model Algebra [7] labels two CSP instances[3] as redundant only if there is a bijective mapping between them. Providing a bijective mapping is a strong condition making their definition of redundancy more restrictive than ours.

Two redundant CSP instances (with respect to a third CSP instance) are *alternative* if their sets of variables are disjoint. We define the *union* of a group of instances as the union of their variables, domains and constraints. The union of alternative CSP instances, redundant with respect to R, represents also the

[3] They say *Models*.

CSP instance R since an extension of either connecting mapping can be used to preserve the solutions. For example, the union of CSP instances 2 and 3 represents CSP instance 1 if function ψ is expanded to take assignments of $rings_1$, $rings_2$ and *switch* but retaining the same mapping used for $rings_1$.

The definition of representation relates two CSP instances. However, it can be extended to include variable and constraint representations.

Definition 2 ***Variable representation.*** *Let x_R be a variable with domain $D(x_R)$. The CSP instance R* ***represents the decision variable*** *x_R if R represents the CSP instance $(\{x_R\}, \{D(x_R)\}, \{\})$.*

Constraint representation. *Let C_R be a constraint over the variables $x_1, \ldots, x_n$ with domains $D(x_1), \ldots, D(x_n)$. The CSP instance R* ***represents the constraint*** *C_R if R represents the CSP instance $(\{x_1, \ldots, x_n\}, \{Dx_1), \ldots, D(x_n)\}, \{C_R\})$.*

These extensions of the concept of representation help to construct representations of sections of a CSP instance. This is needed to compose the *constraint-wise representation* of an instance R.

Definition 3 *Let R be the CSP instance $(X = x_1, \ldots, x_m,\ D_X = D(x_1), \ldots, D(x_m),\ C = \{C_1, \ldots, C_n\})$ and $Y \subseteq X$ be the (possibly empty) set of variables with no constraint in C imposed on them. The CSP instance $R_{cw} = (\bigcup_{1 \leq i \leq n}(\bigcup R_{C_i})) \cup (\bigcup_{x_p \in Y}(\bigcup R_{x_p}))$ is a* ***constraint-wise quasi representation*** *of CSP instance R if*

- *Every R_{C_i} is a non-empty set of CSP instances such that*
  - *The elements of R_{C_i} are pairwise* different *(at least one variable or constraint does not belong to both of them).*
  - *Every $R_{\alpha_i} \in R_{C_i}$ is a constraint representation of C_i.*
  - *For every variable x_k in the constraint C_i there is a CSP instance $R_{\alpha_{ik}} \subseteq R_{\alpha_i}$ representing x_k.*
  - *For every pair of variables x_a, x_b in the constraint C_i, the respective CSP instances $R_{\alpha_i a}$ and $R_{\alpha_i b}$ representing them in R_{α_i}, are* different.
- *For every $x_p \in Y$ let R_{x_p} be a non-empty set of CSP instances such that every $R_{x_{p_\alpha}} \in R_{x_p}$ is a constraint representation of x_p.*

A constraint-wise quasi representation of R is a ***constraint-wise representation*** *of R if it represents R.*

To clarify these definitions attach to CSP instance 1 two constraints which we shall call $C_m(rings)$ and $C_p(rings)$. Attach also to CSP instance 2 some constraint $C'_m(rings_1)$; and to instance 3 some constraint $C'_p(rings_2, switch)$ We call these extensions CSP instances 1a, 2a and 3a respectively (See Figures 4, 5, 6)[4].

[4] To simplify the examples constraints C_m, C'_m, C_p, C'_p remain undefined.

X	$= rings$
$D_X(rings)$	$=$ *multisets (of size nrings) of sets (of maxsize capacity) of 1..nnodes*
C	$= C_m(rings), C_p(rings)$

Fig. 4. CSP-instance 1a of the rings of the Sonet problem.

X_1	$= rings_1$
$D_{X_1}(rings_1)$	$=$ *2-D matrix of Boolean indexed by 1..nrings and 1..nnodes*
C_1	$= \forall\ i \in 1..nrings\ .\ ((\sum j \in 1..nnodes\ .\ rings_1[i,j]) = capacity)$
	$C'_m(rings_1)$

Fig. 5. CSP-instance 2a of the rings of the Sonet problem.

Suppose CSP instance 2a and CSP instance 3a represent constraint $C_m(rings)$ $C_p(rings)$ respectively. Note that each set of instances R_{C_i} contains only one element, and it can be partitioned to obtain the representation of each of the variables the constraint is imposed on. Then the CSP instance $X_{12}, D_{X_{12}}, C_{12})$ composed by the union of instance 2a and 3a (shown in Figure 7) is a constraint-wise quasi representation of CSP instance 1a.

Notice a constraint-wise quasi representation R is a constraint-wise representation when for each variable the same variable representation is used for all the representations composing R. In general, the union of the elements of a constraint-wise quasi representation of an instance does not represent the instance. The main reason, as our example displays is the lack of synchronisation between the alternative representations.

For example for CSP instance 2a $\cup$ 3a that is the case, because the solutions of both representations of *rings* are not connected by any channelling constraint, and for that reason solving the instance will not allow propagation between alternative representations. In a nutshell, any solution of CSP instance $2a \cup 3a$ is an assignment for $rings_1$ representing a different network configuration from the one of $rings_2$ and *switch*, where each configuration satisfies one of the added constraints (C_p, C_m), but not both.

For this reason, it is unlikely we will find a mapping to ensure 2a $\cup$ 3a is a representation of 1a. A solution for the problem instance needs to satisfy both constraints, thus we need to connect both representations of the rings to ensure simultaneous solutions represent the same network configuration and that

X_2	$= rings_2$, *switch*
$D_{X_2}(rings_2)$	$=$ *2-D matrix of 1..nnodes indexed by 1..nrings and 1..capacity*
$D_{X_2}(switch)$	$=$ *2-D matrix of Boolean indexed by 1..nrings and 1..capacity*
C_2	$= \forall\ i \in 1..nrings\ .$ `alldifferent`$(rings_2[i])$
	$C'_p(rings_2, switch)$

Fig. 6. CSP instance 3a of the rings of the Sonet problem.

$$
\begin{array}{ll}
X_{12} & = rings_2,\ switch,\ rings_1 \\
D_{X_{12}}(rings_1) & = \textit{2-D matrix of Boolean indexed by 1..nrings and 1..nnodes} \\
D_{X_{12}}(rings_2) & = \textit{2-D matrix of 1..nnodes indexed by 1..nrings and 1..capacity} \\
D_{X_{12}}(switch) & = \textit{2-D matrix of Boolean indexed by 1..nrings and 1..capacity} \\
C_{12} & = \forall\ i \in 1..nrings\ .\ ((\sum j \in 1..nnodes\ .\ rings_1[i,j]) = capacity) \\
 & \quad C'_m(rings_1) \\
 & \quad \forall\ i \in 1..nrings\ .\ \texttt{alldifferent}(rings_2[i]) \\
 & \quad C'_p(rings_2, switch)
\end{array}
$$

Fig. 7. CSP instance 2a ∪ 3a of the rings of the Sonet problem.

configuration must satisfy both constraints. To transform this constraint-wise quasi representation into a representation we introduce the following channelling constraints:

$$\forall i \in 1..nrings\ . \quad (3)$$

$$((\sum j \in 1..nrings\ .\ \forall k \in 1..nnodes\ .\ \ \exists l \in 1..nnodes\ .\ rings_1[i,k] = rings_1[j,l])$$
$$= \ (\sum j \in 1..nrings\ .\ \exists l \in 1..capacity\ .\ rings_1[i, rings_2[j,l]] \wedge switch[j,l]))$$

Note this channel enforces the synchronisation on each ring, regardless of the order imposed on each representation. Intuitively we assume it's correct because all the solutions of both sides are connected, concordant to the modelling strategy that shaped the representations (in this case the multiset, that is the reason to count the occurrences). We formalise this idea in the following definition of channelling constraints.

Definition 4 *Let R_1 represent R via ψ_1 and R_2 represent R via ψ_2. Let $vars(R_1)$ and $vars(R_2)$ be disjoint sets of variables. The set of constraints C_h is considered a set of* ***channelling constraints between*** *R_1* ***and*** *R_2 if:*

- *For each solution x_1 of R_1 there is at least one total assignment x_2 (of the variables in R_2) such that the composed assignment $x_1 \cup x_2$ satisfies the constraints in C_h. Similarly for each solution x_2 there must be an assignment x_1 such that the composed assignment $x_1 \cup x_2$ satisfies the constraints in C_h.*
- *For all total assignments x_1 and x_2 where the composed assignment $x_1 \cup x_2$ satisfies the constraints in C_h, $\psi_1(x_1)$ and $\psi_2(x_2)$ are either both undefined or take the same value.*

By adding correct channels to the union of a constraint-wise representation of an instance we can now ensure it represents the instance.

Lemma 1 *Let R have two representations R_1 and R_2. Let C_h be a set of channelling constraint between R_1 and R_2. Then $R_1 \cup R_2 \cup C_h$ represents R too.*

Thus, the CSP instance obtained the union of two alternatives and their channels can be used as a representation too.

Lemma 2 *Let R_{cstr} be a constraint-wise quasi representation of R, where only variable x of R has two variable representations R_1 and R_2 in R_{cstr}. Let C_h be the correct channelling constraint between R_1 and R_2. Then, the CSP instance $\bigcup R_{cstr} \cup C_h$ represents R.*

Using Lemma 1 we know $R_1 \cup R_2 \cup C_h$ is a representation of R. Also, whenever R_1 or R_2 are used as representation of R, we could substitute them with $R_1 \cup R_2 \cup C_h$.

This lemma can be extended to cover three or more alternative representations of numerous variables.

The next section describes refinement, an automatic modelling technique implemented in modelling system Conjure [8] in terms of producing constraint-wise quasi representations.

4 Refinement

An automatic modelling tool should assist the user by performing some or all the tasks modellers usually do. In this section we discuss automatic modelling tools that produce constraint models when given a problem specification.

Depending on the language they accept, decisions variables can take diverse domains. In Essence, the input language of the Conjure system [8], decision variables can take domains of sets, multisets, relations and partitions. Some other systems support a more reduced set of domains variables can take. For example, variables in ESRA [9] can take domains of relations over integer domains, while in $\mathcal{F}$ (the language of Fiona) [10] we can specify decision variables whose domain is composed of integer functions, integers and sets of integers. The domains of the variables in Fiona and ESRA cannot be compositional, for example, relations of relations. Due to this lack of domain compositionality the number of transformation cases Fiona and ESRA need to deal with is reduced, allowing them to perform the modelling task with an engine that simply selects and apply the adequate replacements of sections of the given specification.

Unlike Fiona and ESRA, Conjure handles arbitrarily compound domains and for that reason Conjure needs a more powerful engine to transform specifications. This engine is called *recursive refinement*, and in this paper we only discuss the refinement over CSP instances. The refinement performed by Conjure is a generalisation of the refinement restricted to instances explained in this paper. This generalisation is outside of the scope of this paper.

We call a CSP instance *abstract* if it cannot yet be implemented in a current solver. An example of an abstract CSP instance is the one of Figure 1 that represents the network of rings of the Sonet problem with a multiset decision variable. Refinement transforms abstract instances into *concrete* representations that can be implemented directly in a solver. In every step of the refinement this goal is pursued gradually until a representation that can be implemented is produced.

CSP instance 2 and 3 are, according to this notation, concrete representations of CSP instance 1. Concrete instances are often obtained after several refinement

$$
\begin{array}{ll}
X' & = \{\ rings_1\ \} \\
D_{X'}(rings') & = \textit{1-D matrix of sets (of maxsize capacity) of 1..nnodes indexed by 1..nrings} \\
C' & = \emptyset
\end{array}
$$

Fig. 8. CSP-instance $1'$ of the rings of the Sonet problem.

steps, overall when the domain of the variable is compound. For example, CSP instance 1 is initially refined to produce the instance $1'$ of Figure 8.

Given an input CSP instance R, the refinement process generates constraint-wise quasi representations of R. Every constraint-wise representation is generated by independently producing the representations of the constraints and variables of R by means of the recursive application of refinement rules. For example, the *rings* variable of CSP instance 1 is transformed into CSP instance $1'$ of Figure 8 by a rule that transforms multisets into arrays (explicit representation). Then the variable *rings*$'$ is fed to the refinement process to obtain CSP instance 2 of Figure 2 after applying the rule that transforms sets into Boolean arrays (occurrence representation). We can see the refinement process as a generator of a sequence of CSP instances where each instance represents its predecessor. For our example we have two sequences of representations, the one composed by CSP instances 1, $1'$ and 2; and the one composed of CSP instances 1, $1'$ and 3. The following theorem ensures that the last element of a sequence represents the initial CSP instance.

The refinement process as pictured in this explanation appears to work as a cycle that applies replacement rules to specific parts of the instance. The main problem of following an iterative replacement approach is keeping track of the transformations, for example generating alternative representations involves cloning the CSP instance and continue a separate process on the cloned instance.

Instead, the implementation of refinement over CSP instances is a recursive process. In the refinement engine the sequences of representations generated by the refinement process are produced by the recursive application of the *refinement rules*. The purpose of each rule is to reduce the abstraction of the input to return something that can be implemented in a current solver. Many rules are unable to reduce all the abstraction of the input directly, over all when there is a highly compound domain. In several of these cases the transformation between input and output is not direct, then an intermediate CSP instance is constructed inside of the rule needing it. The intermediate CSP instance is similar to the input, but some of the abstraction has been reduced. The rule that generates the intermediate instance handles the rest of the abstraction by calling the refinement process over the abstract parts of the intermediate CSP instance.

To illustrate let's use again our Sonet example. We feed the *rings* variable (CSP instance 1) to the refinement engine, where the **MultisetOfSizeInToExplicit** rule is applied. The **MultisetOfSizeInToExplicit** transforms a multiset variable into its *explicit* representation, that is, a fixed size array of variables where each variable takes the domain of the elements of the multiset. Hence,

the multiset variable *rings* is transformed into the array $rings'$ since the rule composes the intermediate CSP instance $1'$. Afterwards, each of the sets of the $rings'$ variable needs to be refined. At this point two rules can be applied, the **SetOfMaxsizeInToOccurrence** or the **SetOfMaxsizeInToVariableSized-Explicit** rules. The **SetOfMaxsizeInToOccurrence** rule transforms a set variable of bounded size into a Boolean array where the domain of the elements of the set variable is used to index the array. This array composes the so called *occurrence* representation where we set a variable to *True* when the indexing value is in the set. This representation includes constraints to bound the number of elements of the set. For the example, this rule transform the intermediate CSP instance $1'$ into CSP instance 2.

Each constraint is refined separately, and regardless of how it is processed the transformation on variables are always recorded using tags called *representation annotations*. These tags do not contain any information of the rule used for the generation but they contain the variables associated and information specifying the sort of representation constructed by the rule for an specific domain.

Recall that the refinement system returns a set of representations for there is often more than one rule that can be applied each time. Therefore, each alternative way of refining a variable must be given a unique **name**. For non-recursive rules the annotation links directly the input to the output variables. Otherwise, the annotation added by the rule connects the input variable with the variables of the intermediate CSP instance. During the refinement of the intermediate representation more annotations are added for the system in each call to a recursive rule, the last ones connecting the final representations. By ordering the annotations using the variables they associate, we can create sequences of annotations from the input to its final representations. In essence, these sequences detail the sequence of representations the refinement process produce. For example, for the refinements of CSP instance 2 and 3 of CSP instance 1 we have the respective sequences

$$[\texttt{represent}(exp, rings, rings'),\\ \forall i \in 1..nrings.\texttt{represent}(occ, rings'[i], rings_1[i])] \tag{4}$$

$$[\texttt{represent}(exp, rings, rings'),\\ \forall i \in 1..nrings.\ \texttt{represent}(varexp, rings'[i], (rings_2[i], switch[i]))] \tag{5}$$

Note these sequences are also produced by refining CSP instance 1a into the union of instances 2a and 3a. Any CSP instance R', produced by the refinement process of an input CSP instance R, represents R as long as it does not contains several alternative representations of any of the variables in R. For example, one of the refinements of CSP instance 1a is the union of CSP instances 2a and 3a which is clearly not a representation of instance 1a. As we showed in the previous section, channels need to be introduced to transform this union into a proper representation.

5 Systematic generation of channelling constraints

The generation of the needed channels uses the previously discussed sequences of representation annotations that can be constructed from the refinement. More importantly, two representations of a variable are identical if their sequences of annotations obtained from the refinement each one was produced are alphabetic variants[5].

The constraint-wise quasi representation of a CSP instance may contain two (or more) alternative redundant representations of the same variable. We know that to convert it into a proper representations we need to add the needed channelling constraints. We do so by using the sequence of annotations in a second refinement of the variable with alternative representations. We detail as follows the generation of channels between two alternative representations, but the same strategy can be used for three or more alternatives.

Postprocessing Algorithm of Generation of Channels

1. A dummy variable with the same domain of the input variable[6] is created. For our Sonet example we start by creating *ringsdummy*; a variable whose domains is, as well as the *rings* variable, composed of multisets of sets.
2. We modify one of the sequences of annotations to assign a path conducting the dummy variable to an alternative representation. In our example we change only the sequence of (4) into:

$$\begin{aligned}&[\texttt{represent}(exp, ringsdummy, ringsdummy'),\\ &\qquad\qquad \forall i \in 1..nrings.\\ &\texttt{represent}(varexp, ringsdummy'[i], (rings_2[i], switch[i]))] \end{aligned} \tag{6}$$

3. Refine *input_variable* = *dummy_variable*. We refine a channelling constraint between the input variable and the dummy variable. So far, we have used the equality constraint. It is not difficult to verify this is already a correct channelling constraint between the variables. For the example, we refine *rings* = *ringsdummy*.
4. Equality is a channelling constraint between two variables with the same domain. The refinement of a channelling constraint between two variables produces the representation of the two variables plus the channelling constraints between them. For that reason the equality constraint between two variables is refined. For the example, we refine *rings* = *ringsdummy*. From the set of representations we get after the refinement of the channel is finished we select only the ones whose sequences of annotations match with the previously modified ones. Two final representations of a variable are identical if we can *match* their sequences of annotations obtained from each of the refinement process they were produced. This unification is made if both

[5] The name of the variables heading the sequences must be identical. The rest of the elements of the sequence must be identical except from the name of the variables.

[6] Variable for which the refinement generated the alternative representations.

sequences start by annotating the same input variable to the same representation with exactly the same domain but a different name of variable. For the rest of the annotations pair-wise comparison ensures they are identical but for the names of the variables used. In our example the channel (3) is produced.

As expected, this process is based fully on storing the information that shows the transformations that composed a representation, disregarding the set of rules that were applied. The correctness of this technique of generation of channels is ensured by the following theorem:

Theorem 1 *The Postprocessing Algorithm of Generation of Channels transforms a constraint-wise quasi representation with multiple redundant alternative representations into a constraint wise representation.*

Proof. Follows by Lemmas 1 and 2; and the definition of channelling constraint.

This strategy can be used with several variables with multiple representations.

6 Producing the best alternative for channelling

Consider the channelling constraint (1). Direct implementation of this channel attempts to propagate over an extensive set of constraints that may not give the most efficient pruning. Some of the most common constraints over groups of variables have been identified and implemented with particular propagators called global constraints. There are already several global constraints strongly associated with channelling constraints (see [11]).

In our *n-queens* example, the channel (1) could be expressed with the global constraint[7]:

$$\texttt{inverse}(C, R) \tag{7}$$

The technique of automatic generation of channels presented in the previous section is able to generate the most common global constraints used for channels between two alternative redundant representations of a variable. This production is achieved by including rules in the refinement system that refine the equivalence to the needed global constraint. Notice that these global constraints are always imposed to representations that do not need to be refined again; frequently, variables with uncompound domains.

Another method to improve a channelled representation is the deletion of redundant constraints of variables. One of the most widely known examples of this sort of deletion is the removal of the `alldifferent` constraint in a permutation problem [12], if the primal and the dual representation are combined through channelling constraints[8]. Many well documented deletions like this on can be

[7] The syntax of this constraint may vary from solver to solver.

[8] Note this strategy is useful only when a competitive implementation of the channel is at hand, since enforcing GAC on `alldifferent` is stronger.

included in a postprocessing system working over the generated instances that pattern-match the cases.

The real challenge is to compile a big number of reduction rules, including some representations of variables with compound domains. The main difference between the process of reduction and the refinement is that its rules must account all elements of the instance to validate the reduction, whereas during the refinement every part of the instance is treated as separately. For example in Section 2 the channelling constraint (3) between CSP instances 2 and 3 of the Sonet ring variable differs from the simpler one (1) shown in Section 1. The reason is that the order on which the elements of the multiset are arranged is considered different for each representation. Assuming all the representations of the network of the Sonet problem have the same order we can ensure it is safe to replace constraint (3) with (1). This replacement has several consequences, we discard some solutions of each representation since only the cases with equivalent order satisfy the channel; and for that reason, the constraint that replaces the channel does not satisfy the previous definition of channelling constraint since not all the solutions of both representations are preserved. Nevertheless, the combined representation with the new constraints preserves the solutions of the initial variable. As well as in the case of the construction of the refinement rules, we need to carefully compose these reduction rules to guarantee the preservation of solutions.

7 Conclusions and future work

In this paper, we introduce the definitions of CSP instance, variable and constraint representation. We also introduce the definition of constraint-wise (quasi) representation, a structure helpful to describe the transformation method to generate representations used by the refinement process. We show the cases where this method may fail to produce a correct representation are reduced to the ones presenting alternative representation of the same variable. The solution to ensure the preservation of solutions during the refinement is to introduce channelling constraint to synchronise the alternatives.

In Section 5 we discuss a technique to automatically generate the needed channelling constraints. This technique makes use of the very same refinement process, thus lacking the extra burden of implementing an additional subsystem for the generation. Generating the final (and most efficient) implementation of a channel can be made via refinement or reduction. As discussed in Section 6, common global constraints can be produced by adding the respective refinement rules. For some novel representations some known propagators can be adapted or new ones need to be constructed to suit global constraint for channels of novel representations, for example, the channels for the *variable sized explicit* representation introduced by Jefferson et al [5]. Also, as pointed out in the previous section, most of our future work is related to the generalisation of redundancy reduction rules.

References

1. Frisch, A.M., Martínez-Hernández, B.: The systematic generation of channelling constraints. In: Fourth International Workshop on Modelling and Reformulating Constraint Satisfaction Problems. (2005) 89–101 Held at the 11th International Conference on Principles and Practice of Constraint Programming.
2. Cheng, B.M.W., Lee, J.H.M., Wu, J.C.K.: Speeding up constraint propagation by redundant modeling. In: CP 1996. (1996) 91–103
3. Nadel, B.A.: Representation selection for constraint satisfaction: A case study using n-queens. Expert, IEEE **5** (1990) 16–23
4. Frisch, A.M., Hnich, B., Miguel, I., Smith, B.M., Walsh, T.: Transforming and refining abstract constraint specifications. In: 6th International Symposium on Abstraction, Reformulation and Approximation (SARA). (2005) 76–91 LNAI 3607.
5. Jefferson, C., Frisch, A.M.: Representations of sets and multisets in constraint programming. In: Fourth International Workshop on Modelling and Reformulating Constraint Satisfaction Problems. (2005) 102–116 Held at the 11th International Conference on Principles and Practice of Constraint Programming.
6. Law, Y.C., Lee, J.H.M.: Model induction: A new source of CSP model redundancy. In: Eighteenth national conference on Artificial intelligence, American Association for Artificial Intelligence (2002) 54–60
7. Law, Y.C., Lee, J.H.M.: Algebraic properties of CSP model operators. In: Proceedings of the International Workshop on Reformulating Constraint Satisfaction Problems: Towards Systematisation and Automation (Held in Conjunction with CP-2002). (2002) 57–71
8. Frisch, A.M., Jefferson, C., Martínez-Hernández, B., Miguel, I.: The rules of constraint modelling. In: Nineteenth Int. Joint Conf. On Artificial Intelligence (IJCAI). (2005) 109–116
9. Flener, P., Pearson, J., Ågren, M.: Introducing ESRA, a relational language for modelling combinatorial problems. In Rossi, F., ed.: CP 2003, Springer (2003) 971
10. Hnich, B.: Function Variables for Constraint Programming. PhD thesis, Computer Science Division, Department of Information Science, Uppsala University (2003)
11. Beldiceanu, N., Carlsson, M., Rampon, J.X.: Global constraint catalog. Technical Report T2005:08, Swedish Institute of Computer Science (2005)
12. Walsh, T.: Permutation problems and channelling constraints. In Nieuwenhuis, R., Voronkov, A., eds.: Proceedings of LPAR-2001. LNAI, Springer (2001) 377–391