

# The Systematic Generation of Channelling Constraints

Bernadette Martínez-Hernández and Alan M. Frisch

Artificial Intelligence Group, Dept. of Computer Science, Univ. of York, York, UK

**Abstract.** The automatic modelling tool CONJURE generates CSP models from problem specifications. The generated models may contain several redundant representations of the same specification variable. The consistency between the alternative representations is maintained by imposing channelling constraints. In this paper we present an algorithm that produces correct channelling constraints for the generated models using only the facilities already provided by the CONJURE system.

## 1 Introduction

The multiple (concrete) representations of an (abstract) specification variable produce the alternatives to be found in a combined model. The simultaneous assignments of values of each concrete representation must be constrained to stand for the same value for the abstract variable. Hence, we need a method to maintain the consistency between simultaneous redundant representations. This is achieved by imposing channelling constraints. The systematic generation of representations allows to trace how each of the alternatives arises. This understanding of representations helps to produce the correct channelling constraints. In this paper we introduce an algorithm for the systematic generation of channelling constraints between simultaneous redundant representations (under the CONJURE framework). The basic notions related to modelling, channelling constraints and CONJURE, and an outline for the rest of the paper are given as follows.

Solving a problem using CSP technology requires mapping its informal description (often natural language) into a formal description in a particular formal system adequate for constraint solving. This process is called *modelling*. Modelling is a hard task. In many cases the conceptual gap between the informal problem description and the constraint program is large. We may find that a problem is easily modelled with a variable whose domain is unsupported by current solvers. For example, the Sonet problem (shown in Figure 1) requires assigning nodes to a group of rings. This is easily modelled with a variable (*rings*) whose domain is composed by multisets of sets of elements drawn from an integer range (*Nodes*). Variables with this sort of domain are not supported by any current solver. Modellers must use variables with supported domains to represent and implement variables with unsupported domains. In our example, we can use a two-dimensional matrix of integer variables to represent the variable *rings*.

Often, the transformation from an unsupported variable to a supported one involves the addition of constraints to the model. Following the example, to ensure the soundness of the transformation from *rings* to the two-dimensional matrix we need to impose a group of *allDifferent* constraints, one for each represented set.

In an attempt to reduce the burden of hand-crafted modelling, several systems have been created to automate some of the modelling decisions. One of these systems is CONJURE [1], a system that transforms problem specifications into CSP models. Abstract variables in a specification fed to CONJURE may have domains currently unsupported by solvers. CONJURE uses a set of *refinement rules* to compositionally *refine* the variables (and constraints) into concrete representations that can be implemented in current solvers. We briefly describe CONJURE in Section 2.

Modellers often come up with several alternatives to represent an abstract variable. In Section 3 we discuss the definition of representation. We also present some examples of alternative redundant representations.

Different representations may have different strengths. Good modellers know that combining alternative representations in the same model can improve propagation, among other benefits. To maintain the consistency between these simultaneous alternatives we need to add *channelling constraints* [2] to the model. Section 4 discusses channelling constraints (also called *channels*) between alternative representations.

CONJURE is able to produce multiple redundant representations of the same variable. Also, these different alternatives can be generated simultaneously in the same model. Currently, CONJURE does not produce automatically the channelling constraints to maintain the consistency between the alternative representations. We show in Section 5 that it is possible to generate systematically those channels. More importantly, we can use CONJURE for the generation, that is, we do not require implementing a new subsystem for the generation.

Final remarks and future work details can be found in the last section of this paper.

## 2 Refinement and CONJURE

ESSENCE version one (EV1) [4] is the language used to specify problems fed to CONJURE. Variables in EV1 may have associated domains unsupported by current solvers, for example, multisets, functions, relations and partitions. Unlike other languages such as *F* [5], the domain system of EV1 allows domains to be compound to arbitrary depth, thus providing variables with a domain of sets of integers, sets of sets of integers, and so forth.

The current implemented version of CONJURE refines variables whose (arbitrarily compound) domains can be composed by integers, Booleans, sets or multisets. It is planned to extend this implementation of CONJURE to support all the range of domains allowed by EV1. A full description of the EV1 language

A Sonet communication network comprises a number of rings, each joining a number of nodes. A node is installed on a ring using an ADM. There is a capacity bound on the number of nodes that can be installed on a ring. Each node can be installed on more than one ring. Communication can be routed between a pair of nodes only if both are installed on a common ring. Given the capacity bound and a specification of which pairs of nodes must communicate, allocate a set of nodes to each ring so that the given communication demands are met. The objective is to minimise the number of ADMs used. (This is a common simplification of the full Sonet problem, as described in [3])

```

given      nrings:int, nnodes:int, capacity:int
where     nrings ≥ 1, nnodes ≥ 1, capacity ≥ 1
letting   Nodes be int(1..nnodes)
given     demand:set of set (size 2) of Nodes
find      rings:mset (size nrings) of set (maxsize capacity) of Nodes
minimising  $\sum_{r \in rings} |r|$ 
such that  $\forall pair \in demand. \exists r \in rings. pair \subseteq r$ 

```

**Fig. 1.** ESSENCE specification of the Sonet Problem.

and the performance of CONJURE is given in <http://www.cs.york.ac.uk/aig/constraints/AutoModel/>.

An example of an EV1 specification can be found in Figure 1. Keywords identifying the various statements of the specification are shown in **teletype** font. The integer parameters of the problem *nnodes*, *nrings*, *capacity* and *demand* are declared after the keyword **given**. The restrictions for the parameters of a problem instance are specified after the keyword **where**. Following the keyword **letting**, *Nodes* is declared as a ‘short-cut’ for the integer range **int**(1..*nnodes*). The decision variable *rings* is declared as a multiset of sets of integer numbers, after the keyword **find**. Notice that when declared, each parameter and each variable has its domain attached after the symbol ‘:’. The objective function follows the keyword **minimising** and the constraints of the problem follow the keywords **such that**.

The variable *rings* has an associated compound domain. To refine this variable and others of arbitrarily compound domain, CONJURE performs the refinement process in a compositional manner. That is, it first reduces the multiset layer by refining *rings* into a new variable(s) that represents the multiset. This reduction of layers is performed by the recursive application of the *refinement rules*. All refinement rules output a correct representation of their input, where an input can be a variable or a constraint.

To exemplify the compositional construction carried out in each rule let us explain the **SizedMultiset1** rule shown in Figure 2. This rule accepts as input a variable whose domain is composed by multisets of  $\tau$  ( $\tau$  represents any domain), where each multiset has a restricted size *n* (*n* is a placeholder for any expression). Thus, the variable *rings* matches the input of the **SizedMultiset1** rule and this rule can be applied. The **SizedMultiset1** rule ‘peels off’ the multiset layer of

$$\begin{array}{l|l}
\text{SizedMultiset1 } \rho(X_1.\text{mset}(\text{size } m) \text{ of } \tau) \xrightarrow{\text{ref}} & \\
\left\{ \begin{array}{l} X_1'' \\ \text{represent } X_1 \text{ by } \text{expmset}(X_1') \\ | \\ X_1'' \in \rho(X_1') \end{array} \right\} & X_1' = \text{genSymbol}(X_1, \text{matrix}(\text{indexed by } 1..m) \text{ of } \tau)
\end{array}$$

**Fig. 2.** SizedMultiset1 Rule

the variable  $X_1$ . It does so by generating a new variable with the instruction `genSymbol`. The new variable  $X_1'$  has a matrix domain. To refine the following layer ( $\tau$  in the rule, sets for the variable *rings*) of the new variable, the refinement function  $\rho$  must be called recursively. The function  $\rho$  matches its input with the input of a rule and then applies the rule. After the recursive refinement is finished, the **SizedMultiset1** rule returns  $X_1''$ , a correct representation of  $X_1$ . For the variable *rings* the output depends on the refinement rules used by  $\rho$  to transform the set layer.

Often, there are several refinement rules that can be applied to an expression. In fact,  $\rho$  returns a set of alternative refinements. Hence, by the successive application of refinement rules, CONJURE may transform a high-level specification into various different CSP models. Each of the generated models is a correct representation of the original specification and it can be easily implemented in a current solver. At the moment, CONJURE does not implement any heuristic to select effective models from the alternatives generated.

CONJURE is also able to generate models including multiple alternative constructions related to an abstract variable if this variable appears more than once in the constraints of an EV1 specification. For example, in the Sonet problem CONJURE may generate a model with two simultaneous alternatives for *rings*, one to be used in the objective function ( $\sum_{r \in \text{rings}} |r|$ ), and a very different one for the problem constraint ( $\forall_{\text{pair} \in \text{demand}}. \exists_{r \in \text{rings}}. \text{pair} \subseteq r$ ). It is expected that each of the constructed alternatives represent the same value of the specification variable when all the variables of an implementation of the model are instantiated to values. This consistency between alternatives is maintained through channelling constraints.

The current implementation of CONJURE does not generate the channelling constraints. However, the rules add information tags to the model. Some of these tags annotate the relation between the variables used in the rule. For example, we find in the **SizedMultiset1** rule the *channelling annotation tag* ‘**represent**  $X_1$  by `expmset`( $X_1'$ )’ to express the relation between  $X_1$  and  $X_1'$ . In general, the channelling annotations provide information about the type of representation(s) generated by the rule. A human modeller can interpret the information of these tags and use it to construct the needed channelling constraints. The next section discusses the representations and the channelling annotations used by CONJURE to tag the derivation (and relation) of representations. Channelling constraints are formally defined and discussed in Section 4.

### 3 Representations

An abstract variable whose domain is unsupported by a solver must be implemented with a variable or variables of supported domain and possibly some associated constraints over the implemented variables. During the refinement process, the input of a rule can be a variable or a constraint (with its variables) whereas variables (and their domains) and possibly some constraints compose the output of a refinement rule. In both cases we have that a group of variables and constraints is transformed into another group of variables and constraints. The following definition of *CSP instance* captures the notion of a general unit of transformation.

**Definition 1** A *viewpoint* is a pair  $V = (z, D_Z)$  where  $Z$  is a set of variables and  $D_Z$  is a set containing for every variable  $z \in Z$  an associated domain  $D_Z(z)$  defining the set of possible values of  $z$  (domain).

An *assignment* in  $V$  is a pair  $\langle z, a \rangle$ , which means that variable  $z \in Z$  is assigned the value  $a \in D_Z(z)$ . A *total assignment* is a set of unique assignments for each of the variables in  $Z$ .

A *CSP instance* is a pair  $R = (V, C)$  where  $V$  is a viewpoint and  $C$  is a (possibly empty) set of the constraints. The variables (and domains) of each constraint in  $C$  must be included in  $V$ .

A total assignment of the variables of the CSP instance  $R$  is a *solution* if it satisfies all the constraints in  $R$ .

**Example of CSP instance:** The CSP instance **S1** (*set*) consists of a single variable  $S$  whose domain consists of all sets of size  $n$  whose elements are drawn from the integer range  $a..b$ ; where  $a, b$  and  $n$  are integer numbers such that  $a \leq b$  and  $n > 0$ . The set of constraints in **S1** is empty. These definitions of  $a, b, n, S$  and **S1** are used throughout this document.

Viewpoints are defined and used by Law and Lee in their model induction [6] and model algebra [7]. The concept of CSP instance is very similar to their concept of *model*. The main difference is that CSP instances (and viewpoints) are not required to be associated with a problem (immediately). We do, however, describe the conditions for a CSP instance to be represented by another CSP instance. Intuitively, we aim to transform an initial CSP instance into another one that represents it consistently. Hence, the definition of representation must be strongly related to the preservation of solutions.

**Definition 2**  $R'$  *represents*  $R$  via  $\psi$  if  $R' = (V', C')$  and  $R = (V, C)$  are CSP instances and  $\psi$  is a partial function from the total assignments of the variables in  $V'$  into the total assignments of the variables in  $V$  such that:

- For each total assignment  $x'$  of the variables in  $V'$ ,  $x'$  satisfies the constraints in  $C'$  **if and only if**  $\psi(x')$  is defined and satisfies the constraints in  $C$ ,
- For each assignment  $x$  of the variables in  $V$  satisfying the constraints in  $C$ , there is an assignment  $x'$  of the variables in  $V'$  such that  $\psi(x') = x$ .

The triple  $\langle R, R', \psi \rangle$  specifies **the representation** of  $R$  (under  $R'$  and  $\psi$ ).

**Example of representation:** The array of one dimension of integer variables  $VaE1S$  indexed by  $1..n$  composes the variable section of the CSP instance **E1** (*explicit set*). Each element of the array has an integer domain  $a..b$ . The CSP instance **E1** includes constraints that ensure all the elements of the array  $VaE1S$  are different (*allDifferent*). Let  $\psi_E$  be a function from the assignments of  $VaE1S$  into the assignments of  $S$ . The application of  $\psi_E$  is defined only for arrays whose values are all different. The function  $\psi_E$  returns a set containing the values present in the array. Notice that  $\psi_E$  fulfils all the requisites of Definition 2, hence **E1** represents **S1** via  $\psi_E$ .

The variable representation discussed by Jefferson and Frisch [8] is similar to this definition, however, they do not base representations on CSP instances. In fact, they disregards the constraints in its definition. A stronger definition of the function  $\psi$  defines the equivalence of CSP instances.

**Definition 3**  $R'$  is equivalent to  $R$  via  $\psi$  **if and only if**  $R'$  represents  $R$  via  $\psi$ ,  $\psi$  has an inverse, and  $R$  represents  $R'$  via  $\psi^{-1}$ .

**Examples of equivalent representation:** Any CSP instance  $R$  is equivalent to itself via the identity function. An example more related to sets is the CSP instance **O1** (*occurrence set*) that contains an array of one dimension of Boolean variables  $VaO1S$  indexed by the integer range  $a..b$ . The constraint  $sum(VaO1S) = n$  is the only element of the set of constraints. The function  $\psi_O$  from the assignments of  $VaO1S$  to the assignments of  $S$  is defined only for the assignments satisfying  $sum(VaO1S) = n$ .  $\psi_O$  returns a set with the indexes of  $VaO1S$  where the Boolean value **True** is assigned. It is not difficult to conclude that **O1** represents **S1** via  $\psi_O$  and **S1** represents **O1** via  $\psi_O^{-1}$ . Hence **O1** is equivalent to **S1** via  $\psi_O$ .

The one to one correspondence between assignments assumed by the equivalence is similar to the requirement for redundancy of Law and Lee [6]. Among the representations of CONJURE we find important non-equivalent representations of some CSP instances. For example, the CSP instance **E1** is not equivalent to **S1** (neither to **O1**) but it is still a representation of both. Therefore, we use the definition of representation to describe redundancy, that is, two CSP instances are redundant if they represent the same CSP instance. In the CONJURE framework, two different CSP instances generated by the refinement of the same specification variable are redundant if the generation ensures that they are representations of the specification variable. So far, we have discussed CSP instances without focusing in how these CSP instances were constructed to represent another CSP instance. Let us now focus on the core of this construction, the refinement rules.

Notice in Figure 2 we specify the input of the rule **SizedMultiset1** as only one variable ( $X_1$ ). Similarly, a single constraint can be used as the input of a rule. In fact, a single variable can be seen as the CSP instance consisting only of the variable and the empty set of constraints. In the case of a constraint, the

$$\begin{array}{l}
\text{SizedSet1 } \rho(X_1:\text{set}(\text{size } m) \text{ of } \tau) \xrightarrow{\text{ref}} \\
\left\{ \begin{array}{l}
X_1'' \\
\text{represent } X_1 \text{ by } \text{expset}(X_1') \\
\text{such that } \chi \\
| \\
X_1'' \in \rho(X_1') \\
\chi \in \rho(\text{allDifferent}(X_1'))
\end{array} \right. \left. \begin{array}{l}
X_1' = \text{genSymbol}(X_1, \text{matrix}(\text{indexed by } 1..m) \text{ of } \tau)
\end{array} \right\}
\end{array}$$

Fig. 3. SizedSet1 Rule

CSP instance is composed by the variables and domains of the constraint and the constraint as the only element of the set of constraints.

Also, observe the rule **SizedMultiset1** composes an intermediate CSP. The variable  $X_1'$  is the only variable of this intermediate CSP instance. The intermediate CSP instance contains no constraints and it represents the input  $X_1$ . Furthermore, each CSP instance of  $\rho(X_1')$  represents the CSP instance containing only the variable  $X_1'$ . The following theorem ensures, for the cases where an intermediate CSP instance is generated, that each final CSP instance returned by the refinement rule is a representation of the input CSP instance.

**Theorem 1 (Transitivity)** *If  $R''$  represents  $R'$  via  $\psi'$  and  $R'$  represents  $R$  via  $\psi$  then  $R''$  represents  $R$  via  $\psi \circ \psi'$ .*

*Proof.* Straightforward.

Previously, we introduced the channelling annotations as information tags (attached to the model by the rules of CONJURE) used to indicate the relation between two variables. More formally, they provide information to construct the representation relation between two CSP instances. To illustrate this construction let us use some of the former examples.

The **SizedSet1** rule shown in Figure 3 can be used to generate **E1** when refining the variable  $S$ . The variable  $S$  on its own is seen as the CSP instance **S1**. This rule would introduce the channelling annotation ‘**represent  $S$  by  $\text{expset}(VaE1S)$** ’ stating that  $VaE1S$  is the variable used for the explicit set representation of  $S$ . That means **E1** represents **S1** via  $\psi_E$ . Notice that the function  $\psi_E$  and the *allDifferent* constraints of **E1** are implicitly included in the annotation as part of the representation. If the CSP instance **O1** is generated to represent **S1** using the **SizedSet2** rule (see Figure 4), the channelling annotation ‘**represent  $S$  by  $\text{occset}(VaO1S)$** ’ its attached to the model to indicate  $VaO1S$  as the variable used in the occurrence representation of the set  $S$ . Similarly, **O1** represents **S1** via  $\psi_O$ , where  $\psi_O$  and the constraint  $\text{sum}(VaO1S) = n$  are implicitly included on the annotation as part of the obtained representation. From the refinement rules we can produce a catalogue of channelling annotations expressing the different representations used in the refinement.

$$\begin{array}{l}
\text{SizedSet2 } \rho(X_1:\text{set}(\text{size } m) \text{ of } \tau) \xrightarrow{\text{ref}} \\
\left\{ \begin{array}{l}
X_1'' \\
\text{represent } X_1 \text{ by } \text{occset}(X_1') \\
\text{such that } \omega \\
| \\
X_1'' \in \rho(X_1') \\
\omega \in \rho(\text{sum}(X_1') = m)
\end{array} \right. \left. \begin{array}{l}
X_1' = \text{genSymbol}(X_1, \text{matrix}(\text{indexed by } \tau) \text{ of bool})
\end{array} \right\}
\end{array}$$

Fig. 4. SizedSet2 Rule

## 4 Alternative Representations and Channels

Alternative representations of the same variable remain independent until we add channelling constraints to maintain the consistency between them. There is not a precise definition of channelling constraints, hence we present below a definition based on the discussed framework.

**Definition 4** Let  $P_1 = \langle R, R_1, \psi_1 \rangle$  and  $P_2 = \langle R, R_2, \psi_2 \rangle$  be representations of  $R$ . Let  $\text{vars}(R_1)$  and  $\text{vars}(R_2)$  be disjoint sets of variables. The set of constraints  $C_h$  is considered a set of **channelling constraints between  $P_1$  and  $P_2$**  if:

- For each total assignment  $x_1$  (of the variables in  $R_1$ ) satisfying the constraints in  $R_1$  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 satisfying  $x_2$  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.

This definition extends the definition of channelling constraints between variable representations of Jefferson and Frisch [8]. This definition ensures the correct connection between assignments that represent the same value. It also takes into account that some assignments in a representation may be undefined in another representation.

Let us now show the channels between some of the former examples of CSP instances. The CSP instances **S1** and **E1** represent **S1**. Observe that **S1** represents **S1** via the identity function (called here  $\psi_{id}$ ). The following constraint is a channelling constraint between **S1** and **E1**.

$$[\mathbf{CH}_{S_1E_1}] \quad \forall_{j \in a..b}. (j \in S \Leftrightarrow \exists_{i \in 1..n}. \text{VaE1S}[i] = j)$$

Clearly, for every assignment of  $S$  there is at least one assignment of  $\text{VaE1S}$  such that  $S \cup \text{VaE1S}$  satisfies the channel  $\mathbf{CH}_{S_1E_1}$  and  $\text{VaE1S}$  satisfies the constraints of **E1**. This is also satisfied the other way around. Furthermore, the definition of  $\psi_{id}$  and  $\psi_E$  ensure  $\psi_{id}(S)$  and  $\psi_E(\text{VaE1S})$  are either equal or both

undefined when  $S \cup VaE1S$  satisfies  $\mathbf{CH}_{S1E1}$ .

Similarly, the following constraint is a channelling constraint between  $\mathbf{S1}$  and  $\mathbf{O1}$ .

$$[\mathbf{CH}_{S1O1}] \quad \forall_{j \in a..b}. (j \in S \Leftrightarrow VaO1S[j])$$

Suppose that CONJURE produces a model with both  $\mathbf{E1}$  and  $\mathbf{O1}$  as representations of  $\mathbf{S1}$ . It is not very difficult to see the following channelling constraint to connect the alternative representations  $\mathbf{E1}$  and  $\mathbf{O1}$  is correct according to Definition 4.

$$[\mathbf{CH}_{E1O1}] \quad \forall_{j \in a..b}. (VaO1S[j] \Leftrightarrow \exists_{i \in 1..n}. VaE1S[i] = j)$$

We can join two alternative representations of an initial CSP instance together with their channelling constraints. The following theorem ensures that a CSP instance composed this way is also a representation the initial CSP instance.

**Theorem 2** *Let  $\langle R, R_1, \psi_1 \rangle$  and  $\langle R, R_2, \psi_2 \rangle$  be variable disjoint representations of  $R$ , and  $C_h$  a set of channelling constraints between them. We can compose a function  $\psi$  from  $\psi_1$  or  $\psi_2$  such that the triple  $\langle R, R_1 \cup R_2 \cup C_h, \psi \rangle$  represents  $R$  via  $\psi$ .*

*Proof.* Define  $\psi$  as the extended version of  $\psi_1$ , such that it can be applied to any total assignment  $x$  of the union of variables of  $R_1$  and  $R_2$ . The function  $\psi$  returns the same values the function  $\psi_1$  returns whenever a given assignment satisfies the constraints in  $C_h$ . The function  $\psi$  is undefined otherwise. The definition of  $\psi$  and the correct channelling constraints ensure that every total assignment  $x$  of the union of variables of  $R_1$  and  $R_2$  satisfies the constraints in  $R_1$ ,  $R_2$  and  $C_h$  if and only if the function  $\psi(x)$  is defined. Also using the definitions it is not hard to see each solution of  $R$  is obtained by the application of  $\psi$  some total assignment  $x$ . Hence the CSP instance  $R_1 \cup R_2 \cup C_h$  satisfies the requirements to represent  $R$  via  $\psi$ .  $\square$

Note that  $\langle \mathbf{S1}, \mathbf{E1} \cup \mathbf{O1} \cup \mathbf{CH}_{E1O1}, \psi'_O \rangle$  is a representation of  $\mathbf{S1}$  where  $\psi'_O$  is the extension of  $\psi_O$

The refinement of  $R$  returns a CSP instance  $R'$  and a set of annotations declaring all the intermediate CSP instances representing  $R$  where  $R'$  represents all of them. That is, the annotations describe a sequence of refinements from  $R'$  to  $R$ . We call  $R'$  the *final representation* of  $R$ . In the next section we discuss the generation of channelling constraints between two alternative final (redundant) representations.

## 5 Systematic Generation of Channelling Constraints

We have shown how we can relate two CSP instances using the information provided by the channelling annotations: more importantly, we presented a definition of channelling constraints between representations based on the notions

of CSP instances and representations. The examples of channelling constraints shown were fairly simple; however the generation of channelling constraints based on the annotations becomes challenging for human modellers when the input variables have deeply compound domains. It is not unreasonable to think that the automation of the generation of the channelling constraints can be provided by the refinement system already provided by CONJURE.

Let  $P$  be a specification refined by CONJURE into  $P'$  where the variable  $X$  in  $P$  has two final representations in  $P'$ ,  $X_1$  and  $Y_1$ . Suppose we can use the annotations to force CONJURE to produce only certain representations (with the same domains) for certain variables. That is, we can ask CONJURE to generate specific representations following a path of refinements given by the annotations. Let  $Y$  be a new variable with exactly the same domain of  $X$ . The constraint  $X = Y$  fulfils the definition of channelling constraint between  $X$  and  $Y$ , where both are representations of  $X$ . We can then *re-refine*  $X = Y$  forcing the  $X$  to refine into  $X_1$  and the  $Y$  to refine into  $Y_1$ . Such refinement produces a representation of the channelling constraints between  $X_1$  and  $Y_1$ . Hence, we can compose a representation of  $X$ ,  $\langle X, X_1 \cup Y_1 \cup \rho_{X_1, Y_1}(X = Y), \psi_{X_1} \rangle$ , where  $\rho_{X_1, Y_1}(X = Y)$  is the conditioned refinement of  $X = Y$ . We call this algorithm of generation the *post-processing algorithm*.

**Theorem 3** *If the rules of CONJURE produce only representations of their inputs then the **post-processing algorithm** generates equivalent (correct) models.*

*Proof Sketch:* If each specification variable has only one representation in a CONJURE generated model we can ensure that the produced model is a representation of the specification problem if the refinement always returns representations.

Let us restrain CONJURE to produce only models with one representation of each variable (if any). Suppose we have a variable  $A$  with two occurrences in the constraints of the specification  $M$ . Let  $M'$  be the model  $M$  with a new variable  $A'$  of the same domain of  $A$ , the constraint  $A = A'$ , and one of the occurrences of  $A$  substituted by  $A'$ . It is not hard to show that  $M'$  represents  $M$  as long as all the rules preserve the representation property. Then, a model  $M''$  obtained by the post-processing algorithm representing  $M'$  represents  $M$  too. In fact we can prove that there is a model  $M'$  which is equivalent (if not identical) to each  $M''$  generated by the post-processing algorithm.  $\square$

We show now an example of the post-processing algorithm with a modified version of the variable *rings* of the Sonet problem. Let us define the domain of the variable *rings* as all the multisets of size *nrings*, where each of the multisets has for elements sets of size *capacity* of the integer range *Nodes*. Suppose we apply the **SizedMultiset1** rule to *rings*, then, a new variable *rings'* is introduced. The elements of the domain of this variable *rings'* are arrays of sets of integers. Each array is indexed by the integer range  $1..n$ . The variable *rings'* is used as input for the recursive call to the refinement function. This call will eventually use either the **SizedSet1** rule or the **SizedSet2** rule.

The application of the **SizedSet1** rule produces the variable *rings'*<sub>1</sub> whose domain elements are two dimensional matrices of integer variables. All matrices

$$\begin{array}{l}
\mathbf{SizedMultisetEquality1} \ \rho(X_1:\mathbf{mset}(\text{size } m) \text{ of } \tau = X_2:\mathbf{mset}(\text{size } m) \text{ of } \tau) \xrightarrow{\text{ref}} \\
\left\{ \begin{array}{l} \phi \\ \text{represent } X_1 \text{ by } \mathit{expmset}(X'_1) \\ \text{represent } X_2 \text{ by } \mathit{expmset}(X'_2) \\ | \\ \phi \in \rho(\forall_{i \in 1..m}. \exists_{j \in 1..m}. X'_1[i] = X'_2[j]) \end{array} \right. \left. \begin{array}{l} X'_1 = \mathit{genSymbol}(X_1, \mathit{matrix}(\text{indexed by } 1..m) \text{ of } \tau) \\ X'_2 = \mathit{genSymbol}(X_2, \mathit{matrix}(\text{indexed by } 1..m) \text{ of } \tau) \end{array} \right\} \\
\}
\end{array}$$

**Fig. 5.** SizedMultisetEquality1 Equality Rule

$$\begin{array}{l}
\mathbf{SizedSetEquality1} \ \rho(X_1:\mathbf{set}(\text{size } m) \text{ of } \tau = X_2:\mathbf{set}(\text{size } m) \text{ of } \tau) \xrightarrow{\text{ref}} \\
\left\{ \begin{array}{l} \phi \\ \text{represent } X_1 \text{ by } \mathit{expset}(X'_1) \\ \text{represent } X_2 \text{ by } \mathit{occset}(X'_2) \\ \text{such that } \chi \wedge \omega \\ | \\ \phi \in \rho(\forall_{j \in \tau}. X'_2[j] \Leftrightarrow \exists_{i \in 1..m}. X'_1[i] = j) \\ \chi \in \rho(\mathit{allDifferent}(X'_1)) \\ \omega \in \rho(\mathit{sum}(X'_2) = m) \end{array} \right. \left. \begin{array}{l} X'_1 = \mathit{genSymbol}(X_1, \mathit{matrix}(\text{indexed by } 1..m) \text{ of } \tau) \\ X'_2 = \mathit{genSymbol}(X_2, \mathit{matrix}(\text{indexed by } \tau) \text{ of } \text{bool}) \end{array} \right\} \\
\}
\end{array}$$

**Fig. 6.** SizedSetEquality1 Equality Rule

of this domain must be indexed by the integer ranges  $1..nrings$  and  $1..capacity$ . The channelling annotations introduced by this refinement are ‘**represent rings** by  $\mathit{expmset}(\mathit{rings}'_1)$ ’ and ‘ $\forall_{i \in 1..nrings}. \mathbf{represent rings}'_1[i]$  by  $\mathit{expset}(\mathit{rings}'_1[i])$ ’.

On the other hand, the application of the **SizedSet2** rule produces the variable  $\mathit{rings}'_2$  whose domain is integrated by two-dimensional matrices of Boolean variables. All matrices of this domain must be indexed by the integer ranges  $1..nrings$  and  $1..nodes$ . The channelling annotations introduced by this refinement are ‘**represent rings** by  $\mathit{expmset}(\mathit{rings}'_2)$ ’ and ‘ $\forall_{j \in 1..nrings}. \mathbf{represent rings}'_2[j]$  by  $\mathit{occset}(\mathit{rings}'_2[j])$ ’.

If these two representations,  $\mathit{rings}'_1$  and  $\mathit{rings}'_2$  are generated in the same model we need to construct the correct channelling constraint. For the post-processing algorithm we introduce the new variable  $\mathit{rings}$  to be restricted to refine into  $\mathit{rings}'_2$ . The restricted refinement of the constraint  $\mathit{rings} = \mathit{rings}$  uses the **SizedMultisetEquality1** rule (Fig. 5) first, and then, by means of the recursive calls to the refinement function the **SizedSetEquality1** rule (Fig. 6) is applied. Finally, the following channelling constraint is obtained.

$$\begin{array}{l}
[\mathbf{CH}_{\mathit{rings}}] \qquad \forall_{i \in 1..nrings}. \exists_{j \in 1..nrings}. \forall_{k \in 1..nodes}. \\
(\mathit{rings}'_1[i, k] \Leftrightarrow \exists_{l \in 1..capacity}. \mathit{rings}'_2[j, l] = k)
\end{array}$$

## 6 Conclusion and Future Work

We present in this paper a framework relating representations of the same variable in different levels of abstractions. The generation of channelling constraints presented is based on the sequence of representations of an specification variable.

The algorithm presented, based on generalisations over CSP instances, reduce the problem of automatic generation of channelling constraints to the problem of proving the correctness of the refinement rules of CONJURE. This method of generation may work not only in the CONJURE system, but also in any system transforming CSP instances with an engine sufficient enough to transform the constraint ( $A = A'$ ).

It is important to notice the post processing algorithm can be generalised to produce the channels between three or more representations of the same specification variable. We may even modify the strategy of generation and produce only some channelling constraints instead of all the multiple channels between these representations.

The work is far from being complete. The conditions and performance of the generation of channels for non (totally) redundant representations are not included in this paper. Also, correct channels may need post-processing for a better reading of the user and/or to generate efficient code for a CSP solver. What is more, due to the nature of the refinement process we may generate several valid alternative channelling constraints. This alternative generation increases the complexity of the model selection task.

## 7 Acknowledgements

We thank to Ian Miguel and Chris Jefferson for the fruitful discussions and comments of this work.

## References

1. Frisch, A.M., Jefferson, C., Martínez-Hernández, B., Miguel, I.: The rules of constraint modelling. In: Nineteenth Int. Joint Conf. on Artificial Intelligence. (2005)
2. Cheng, B.M.W., Choi, K.M.F., Lee, J.H.M., Wu, J.C.K.: Increasing constraint propagation by redundant modeling: An experience report. *Constraints* **4** (1999) 167–192
3. Frisch, A.M., Hnich, B., Miguel, I., Smith, B.M., Walsh, T.: Transforming and refining abstract constraint specifications. In: Symposium on Abstraction, Reformulation and Approximation (SARA). (2005)
4. Frisch, A.M., Grum, M., Jefferson, C., Martínez-Hernández, B., Miguel, I.: The Essence of Essence. In: International Workshop on Modelling and Reformulating Constraint Satisfaction Problems. (2005) Held at the 11th International Conference on Principles and Practice of Constraint Programming.
5. Hnich, B.: Function Variables for Constraint Programming. PhD thesis, Computer Science Division, Department of Information Science, Uppsala University (2003)

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 Shorter paper in CP-2002.
8. Jefferson, C., Frisch, A.M.: Representations of sets and multisets in constraint programming. In: International Workshop on Modelling and Reformulating Constraint Satisfaction Problems. (2005) Held at the 11th International Conference on Principles and Practice of Constraint Programming.