A Note on Redundant Rules in Rule-based Constraint Programming

Sebastian Brand

CWI, P.O. Box 94079, 1090 GB, Amsterdam, The Netherlands

Abstract. Constraint propagation can often be described conveniently in a rule-based way. Propagation is then fixpoint computation with rules. It is preferable to have a minimal set of rules, in particular for the typical case when no intelligent strategy guides the fixpoint computation. We propose an intuitive criterion for redundancy of a rule, and describe an effective test for a class of rules. The usefulness is shown by relating it to two important approaches to automatic rule generation.

1 Introduction

Constraint propagation is an essential ingredient for successful solving of constraint satisfaction problems. A general approach of describing the propagation process is to use a rule-based language such as ELAN and `CHR` [BKK+96,Frü98]. The rules are applied exhaustively in a fixpoint computation, and in that way constraint propagation takes place.

A difficulty lies in obtaining suitable propagation rules. In the recent years a series of articles has dealt with this issue [AM01,RM00,AR00,AR01]. These approaches take constraint definitions, extensionally by stating all valid tuples or intensionally by a constraint logic program, and generate automatically a set of propagation rules.

Example. For the constraint $and(x, y, z)$ that describes $x \wedge y = z$ for variables $x, y, z \in \{0, 1\}$ we may generate the rules $x = 0 \rightarrow z = 0$; $y = 1, z = 0 \rightarrow x = 0$; etc. Applying them in any order to $and(x, 0, z)$ until no rule is applicable anymore leads to the fixpoint $and(x, 0, 0)$. □

Typically there are many valid rules for a constraint, but not all are interesting for a constraint solver. A rule that is less general than another one can be discarded since its effect is subsumed. Rule generation therefore needs a concept of redundancy. The above-mentioned approaches, however, treat redundancy almost exclusively by reference to a single other rule, e. g., $x = 0 \rightarrow z = 0$ subsumes $x = 0, y = 0 \rightarrow z = 0$. We argue that a wider concept is appropriate: redundancy with respect to a set and referring to the fixpoints of the set.

Example. For one constraint (`and11`), the rule generation algorithm of [AM01] computes a set of 4656 constraint propagation rules. It does not contain redundancies by one other rule. Removing from this set all rules that are redundant by the remaining set yields 393 rules. This minimal rule set achieves the same constraint propagation. □

We proceed by introducing a simple and intuitive definition of redundancy in the general framework of fixpoint computation over functions [Apt99]. This is instantiated to the case of functions in rule form, which subsequently allows a connection between a few requirements on rules and a simple, effective redundancy test. We establish the usefulness of this by considering two important cases of automatic rule generation where automatic redundancy detection is essential: membership rules [AM01] and a class of `CHR` propagation rules [Frü98,AR00]. Both fit in our framework, and hence our wider redundancy criterion can lead to reduced sets of rules.

2 Redundancy with Respect to Fixpoints

2.1 Preliminaries

We begin the presentation by recalling, concisely, the chaotic iteration framework of [Apt99]. We deal with a partially ordered set and functions defined on it. It is shown below how constraint propagation can be explained in this system.

Definition 1. *Let* $(D, \sqsubseteq)$ *be a partial order. Let* F *be a set of functions* $\{f_1, \ldots, f_n\}$ *on* D*. A function* f *is called*

- inflationary *if* $\quad d \sqsubseteq f(d) \quad$ *for all* d*,*
- monotonic *if* $\quad d \sqsubseteq e \;\Rightarrow\; f(d) \sqsubseteq f(e) \quad$ *for all* d, e*.* □

We are interested in fixpoints of such functions. By a chaotic iteration of F we mean an infinite sequence $d_0, d_1, \ldots$ such that $d_{i+1} = f(d_i)$ for some $f \in F$, and in which each $f \in F$ is applied infinitely often. An iteration stabilises at some e if an index k exists such that for all $i \geqslant k$ we have $d_i = e$. A simple consequence of the Stabilization Lemma in [Apt99] is the following.

Lemma 2. *If all functions in* F *are inflationary and monotonic and the corresponding partial order* $(D, \sqsubseteq)$ *is finite then every chaotic iteration of* F *starting in* d *stabilises at the least fixpoint that is greater than or equal to* d*.* □

The least fixpoint greater than or equal to a given point can thus be effectively computed by repeatedly applying the functions in any order until stabilisation is obtained.

2.2 Redundant Functions

The cost of such a fixpoint computation depends on the number of functions involved, in particular in absence of a good strategy to select functions. It is therefore interesting to identify functions that are not needed for computing fixpoints. We define redundancy accordingly.

Definition 3. *A function* f *is redundant with respect to a set* F *of functions if the sets of fixpoints of* F *and* $F \cup \{f\}$ *are equal.* □

An equivalent formulation is to say that f is redundant if every fixpoint of F is a fixpoint of f as well.

3 Rules

The type of functions we are interested in are rules of the form $b \to g$, where b is a condition and the conclusion g a function. The condition is evaluated on the elements of D, we write $Holds(b,d)$ if b is true on d. The application of rules is defined by

$$(b \to g)(d) := \begin{cases} g(d) & \text{if } Holds(b,d) \\ d & \text{otherwise} \end{cases}$$

We are now interested in rules with particular properties.

Definition 4. *A condition b is*

- monotonic *if*

$$Holds(b,d) \ \wedge \ d \sqsubseteq e \quad \Rightarrow \quad Holds(b,e)$$

  for all d, e.
- precise *if a point w exists that satisfies $Holds(b,w)$, and also*

$$Holds(b,d) \quad \Rightarrow \quad w \sqsubseteq d$$

  for each d. We call w, which is unique, the precision witness *of b.*

A function g is stable *if*

$$g(d) \sqsubseteq e \quad \Rightarrow \quad g(e) = e$$

holds for all d, e □

The following connections between properties of a rule and properties of its body function are easy to prove.

Lemma 5. *Consider a rule $b \to g$. If g is inflationary then so is $b \to g$. If b and g are monotonic then so is $b \to g$.* □

Note also that if g is a function such that d and $g(d)$ are comparable for all d, then stability of g implies inflationarity.

3.1 Redundant Rules

The above properties are useful as they allow a simple test for redundancy.

Theorem 6. *Consider a rule $r = b \to g$ such that its condition b is monotonic and precise with witness w, and its body function g is stable. Let e be the least fixpoint of F greater than or equal to w. The rule r is redundant with respect to F if $g(e) = e$ holds.*

Proof. We show that $g(e) = e$ implies that an arbitrary fixpoint d of F is a fixpoint of r, by a case distinction on the condition.

b holds for d: We have $w \sqsubseteq d$ from the precision of b. Also, $w \sqsubseteq e \sqsubseteq d$ since e is the least fixpoint greater than or equal to w. From $e \sqsubseteq d$, $g(e) = e$, and stability of g we conclude $g(d) = d$ and finally $r(d) = (b \to g)(d) = d$.

b does not hold for d: Here $r(d) = (b \to g)(d) = d$ holds immediately. □

This test is interesting as it needs to compute only one fixpoint of F instead of all fixpoints. It is effective if

- the precision witness is accessible,
- $g(e) = e$ can be decided, and
- fixpoint computations are effective (cf. Lemma 2).

Definition 7. *A set of rules F is said to be* minimal with respect to redundancy *(or just minimal), if no $r \in F$ is redundant with respect to $F - \{r\}$.* □

Minimal sets of rules can of course be obtained by a simple bounded loop: choose an untested rule, test whether it is redundant and remove it from the current set if it is. In general, however, the obtained minimal set depends on the order in which the rules were tested; see the example later. The questions for

- a total preference between two rule sets, and/or
- a rule selection strategy approximating such a preference

are open. For our experiments we used a selection strategy that prefers to keep rules that are deemed to be computationally less costly.

Partial redundancy. We point out that a rule $r = b \to g$ with multiple conclusions $g = g_1, \ldots, g_n$ (describing a conjunction) can be understood as the collection $b \to g_1, \ldots, b \to g_n$ of rules with identical condition, and vice versa. The respective fixpoints, and rule properties, are the same. We regard here the different incarnations as equivalent. If a rule as a whole is not redundant it might be partially redundant, that is, some sub-rules of its decomposition are redundant; or in other words, some part of its conclusion is redundant.

3.2 Redundancy by One Rule

We examine a special case involving only two rules, one 'stronger' than the other.

Corollary 8. *Consider a rule $r = b \to g$ in a rule set F, and a rule $r' = b' \to g'$ not in F that satisfies the requirements mentioned in Theorem 6, and moreover is such that g' is inflationary. Assume that*

$$\mathit{Holds}(b', d) \ \Rightarrow\ \mathit{Holds}(b, d) \qquad\qquad \textit{and} \qquad\qquad g'(d) \ \sqsubseteq\ g(d)$$

holds for all d. Then r' is redundant with respect to F.

Proof. Let e be the least fixpoint of F greater than or equal to to the witness w' of b'. We show that $g'(e) = e$, which entails the desired result by Theorem 6.

We have $\mathit{Holds}(b', w')$ and monotonicity of b', thus also $\mathit{Holds}(b', e)$. The first requirement above entails now that $\mathit{Holds}(b, e)$. We know for the fixpoint e that $e = r(e)$, and with $\mathit{Holds}(b, e)$ also $e = g(e)$. With the second requirement we conclude $g'(e) \sqsubseteq e = g(e)$, but g' is also inflationary: $e \sqsubseteq g'(e)$. Hence, $g'(e) = e$. □

4 Membership Rules

In [AM01] a specific type of rules, called membership rules, is presented. These rules are the output of a rule generation algorithm whose input is an extensionally defined constraint. In a fixpoint computation they enforce hyper-arc consistency. The generated rule set contains only *minimal rules* in the sense that no rule 'extends' another. As it turns out, the redundancy notion proposed here can be applied to reduce the generated sets of membership rules even more.

Constraint satisfaction problems. Consider a sequence $X = x_1, \ldots, x_n$ of variables with associated domains $D_1, \ldots, D_n$. By a constraint C on X we mean a subset of $D_1 \times \ldots \times D_n$. Given an element $d = d_1, \ldots, d_n$ of $D_1 \times \ldots \times D_n$ and a subsequence $Y = x_{i_1}, \ldots, x_{i_\ell}$ of X we denote by $d[Y]$ the sequence $d_{i_1}, \ldots, d_{i_\ell}$.

A constraint satisfaction problem (CSP), consists of a finite sequence of variables X with respective domains $\mathcal{D}$, together with a finite set $\mathcal{C}$ of constraints, each on a subsequence of X. We write it as $\langle \mathcal{C}; x_1 \in D_1, \ldots, x_n \in D_n \rangle$, where $X = x_1, \ldots, x_n$ and $\mathcal{D} = D_1, \ldots, D_n$. By a solution to $\langle \mathcal{C}; x_1 \in D_1, \ldots, x_n \in D_n \rangle$, we mean an element $d \in D_1 \times \ldots \times D_n$ such that for each constraint $C \in \mathcal{C}$ on a sequence of variables X we have $d[X] \in C$.

Partial Ordering. With a CSP $\langle \mathcal{C}; x_1 \in D_1, \ldots, x_n \in D_n \rangle$ we associate now a specific partial order: the Cartesian product of the partial orders $(\mathcal{P}(D_i), \supseteq)$ Hence this order is of the form

$$(\mathcal{P}(D_1) \times \ldots \times \mathcal{P}(D_n), \supseteq)$$

where $\supseteq$ is the Cartesian product of the reversed subset order. The elements of this partial order are sequences $(E_1, .., E_n)$ of respective subsets of $(D_1, \ldots, D_n)$

The membership rules of [AM01] acceptable for a specific CSP over variables $X = x_1, \ldots, x_n$ with finite domains are as follows. The rules have the form

$$y_1 \in S_1, \ldots, y_k \in S_k \ \rightarrow \ z_1 \neq a_1, \ldots, z_m \neq a_m$$

where $y_1, \ldots, y_k$ are pairwise different variables from X, and $S_1, \ldots, S_k$ are subsets of the respective variable domains. Also the $z_1, \ldots, z_m$ are pairwise different variables from X, while $a_1, \ldots, a_m$ are elements of the respective variable domains. If each S_i is a singleton then a membership rule is called *equality rule.*

The computational interpretation of a membership rule is: if the current domain of the variable y_i is included in the set S_i for all $i \in [1..k]$, then remove the element a_j from the domain of z_j for all $j \in [1..m]$.

Condition test and body application. We define the atomic case

$$Holds(x_i \in S, E) \ := \ E_i \subseteq S$$

where $E = (E_1, \ldots, E_n)$ is an element of the partial order for a CSP with variables $x_1, \ldots, x_n$. A sequence of conditions holds if each atomic conditions does.

Applying a rule body $z_1 \neq a_1, \ldots, z_m \neq a_m$ takes place stepwise, with

$$(x_i \neq a)(E) \ := \ E_i \setminus \{a\}$$

defining the atomic case.

Properties. The properties of rules and functions hold for membership rules. To see this consider a CSP $\langle \mathcal{C}; x_1 \in D_1, \ldots, x_n \in D_n \rangle$ and an associated rule

$$r \quad = \quad b \to g \quad = \quad y_1 \in S_1, \ldots, y_k \in S_k \ \to \ z_1 \neq a_1, \ldots, z_m \neq a_m \ .$$

Precise condition: We define

$$S'_i = \begin{cases} S_j & \text{if } x_i \text{ occurs as } y_j \text{ in } b \\ D_i & \text{if } x_i \text{ does not occur in } b. \end{cases}$$

Consider now the element $E = (S'_1, \ldots, S'_n)$ of the partial order. Observe that condition b holds at E. Moreover, it holds only for elements $E' \subseteq E$. Hence, E is a precision witness, and b is precise.

Monotonic condition: It is easy to see that this holds.

Monotonicity, inflationarity, stability of body: All true, by trivial proofs.

The redundancy test of Theorem 6 is therefore applicable to sets of membership rules. Note in particular that the required precision witness can be extracted immediately from the the rule and the CSP at hand.

4.1 An Example: Redundant Membership Rules

We illustrate a number of issues by way of example. Suppose the constraint $c(x, y, z, u)$ is defined by the three tuples of Table 1. The underlying domain for all its variables is $\{0, 1\}$, hence the induced corresponding partial order is $(\{\, \{0,1\} \times \{0,1\} \times \{0,1\} \times \{0,1\}, \ldots, \varnothing \times \varnothing \times \varnothing \times \varnothing\} \ \supseteq)$. The algorithm of [AM01] produces a set of 11 membership rules (in fact equality rules).

The presence of rule 11, $c(1, y, z, 1) \to y \neq 1$, states that if $c(x, y, z, u)$ then it is correct to conclude from $x = 1$ and $u = 1$ that $y \neq 1$ (validity), and furthermore that neither $x = 1$ nor $u = 1$ suffices for this conclusion (minimality).

Let us examine a fixpoint of these rules: we are interested in the smallest fixpoint greater than or equal to $E_1 = \{1\} \times \{0, 1\} \times \{0, 1\} \times \{1\}$ which corresponds to the CSP in the preceding paragraph. Suppose rule 11 is considered. Its application yields $E_2 = \{1\} \times \{0\} \times \{0, 1\} \times \{1\}$ from where rule 4 leads to $E_3 = \{1\} \times \{0\} \times \{0\} \times \{1\}$. This is indeed a fixpoint since each rule either does not apply or its application results again in E_3.

Redundant rules. A second possible iteration from E_1 that stabilises in E_3 is by rule 5 succeeded by rule 10. Rule 11 can be applied at this point but its body effects no change on E_3; E_3 is a fixpoint of all rules including rule 11. We conclude that rule 11 is redundant - we just performed the test of Theorem 6.

The process of identifying redundant rules can then be continued with the rule set $\{1, \ldots, 10\}$. A possible outcome is depicted in Table 1 where redundant parts of rules are underlined. From the 20 initial atomic conclusions 13 remain, thus we find here a reduction ratio of 65%.

c	x	y	z	u
	0	1	0	1
	1	0	0	1
	1	1	1	0

$$
\begin{array}{lcll}
c(x,y,z,0) & \rightarrow & x \neq 0, y \neq 0, z \neq 0 & (1) \\
c(x,y,1,u) & \rightarrow & u \neq 1, \underline{x \neq 0, y \neq 0} & (2) \\
c(0,y,z,u) & \rightarrow & u \neq 0, y \neq 0, \underline{z \neq 1} & (3) \\
c(x,0,z,u) & \rightarrow & u \neq 0, x \neq 0, \underline{z \neq 1} & (4) \\
c(x,y,z,1) & \rightarrow & z \neq 1 & (5) \\
c(x,y,0,u) & \rightarrow & u \neq 0 & (6) \\
c(1,1,z,u) & \rightarrow & u \neq 1, \underline{z \neq 0} & (7) \\
c(x,1,0,u) & \rightarrow & x \neq 1 & (8) \\
c(x,1,z,1) & \rightarrow & \underline{x \neq 1} & (9) \\
c(1,y,0,u) & \rightarrow & y \neq 1 & (10) \\
c(1,y,z,1) & \rightarrow & \underline{y \neq 1} & (11)
\end{array}
$$

Table 1. Definition and membership rules for the constraint c

Nondeterminism in rule removal. Consider the justification for the redundancy of rule 11, and observe that rule 11 has no effect since rule 10, which has the same body, was applied before. Suppose now that the process of redundancy indentification is started with rule 10 instead of rule 11. This results in rule 10 being shown redundant, with a relevant application of rule 11.

Note moreover, that one of the rules 10,11 must occur in *any* reduced rule set since their common body $y \neq 1$ occurs in no other rule. It seems difficult to find a good criterion which of the rules 10,11 this should be.

4.2 Rule Generation

The algorithm of [AM01] that generates the set of rules from a given constraint can straightforwardly be abstracted to:

1. collect all (syntactically and semantically) valid membership rules
2. discard those that *extend* another present rule

The membership rule $b' \rightarrow g'$ *extends* the rule $b \rightarrow g$ if $g' = g$ and (using our terminology) $\mathit{Holds}(b', d) \rightarrow \mathit{Holds}(b, d)$. If a rule does not extend another one then it is called *minimal*.

The minimality requirement is weaker than that of Corollary 8. A non-minimal rule is therefore a special case of the rule being redundant. We conclude that a generated set of membership rules has the same set of fixpoints as the set of all valid membership rules, and point out that the generated set can be expected not to be minimal, even though it only contains minimal rules.

5 Propagation Rules

The concept of a CSP in the section on membership rules emphasises domains as a channel through which constraint propagation takes place. We now take a different position: we understand a CSP as just a set of constraints. Domains are unary constraints.

5.1 Abstract Propagation Rules

A propagation rule in this framework describes that some constraints should be added to the problem if some other constraints are present. This can be viewed as the generation of simpler, redundant constraints. We show in the following that also this model fits into the partial order framework. The major goal we pursue is to make a connection between the partial order framework, and propagation rules as used in the `CHR` language and in the work [AR00] on automatic rule generation.

Constraints. A constraint is an atomic formula on variables. The language $\mathcal{L}$ is defined as consisting of all constraints over a finite set of variables and a finite set of constraint symbols. A CSP is a set of constraints.

Partial Ordering The construction of the corresponding partial order $(D, \sqsubseteq)$ is simple. We define it as

$$(\mathcal{P}(\mathcal{L}), \subseteq) \ .$$

Rules. Propagation rules are of the form $C_L \to C_R$ where C_L and C_R are sets of constraints of $\mathcal{L}$.

Condition test and body application. Suppose $C_L \to C_R$ is a propagation rule, and $E \in D$ is a set of constraints. The evaluation of the condition C_L on E is defined by

$$\mathit{Holds}(C_L, E) \ := \ C_L \subseteq E$$

while the application of the body C_R is

$$C_R(E) \ := \ E \cup C_R$$

This explains the application $(C_L \to C_R)(E)$.

Properties. The rule properties introduced above hold for propagation rules. We give below the proof obligations for an arbitrary rule $C_L \to C_R$. The proofs themselves,trivial set theory, are omitted.

Precise condition: Take as the witness C_L. To show: $C_L \subseteq E \;\Rightarrow\; C_L \subseteq E$
Monotonic condition: $E_1 \subseteq E_2 \;\Rightarrow\; E_1 \cup C_R \subseteq E_2 \cup C_R$
Monotonic body: $C_L \subseteq E_1 \wedge E_1 \subseteq E_2 \;\Rightarrow\; C_L \subseteq E_2$
Stability of body: $C_L \cup E_1 \subseteq E_2 \;\Rightarrow\; C_L \cup E_2 = E_2$
Inflationarity of body: $C_L \cup E_1 \subseteq E_2 \;\Rightarrow\; C_L \cup E_2 = E_2$

Abstract propagation rules are amenable to the redundancy test of Theorem 6.

5.2 A Class of CHR Propagation Rules

`CHR` [Frü98] is a high-level rule-based language especially designed for writing constraint solvers. It defines propagation rules that can add constraints, and simplification rules that replace constraints by simpler ones. Both types can have an extra guard on the variables. The class of guard-free `CHR` propagation rules in which the body does not introduce new variables and is interpreted as a set of constraints, however, corresponds to abstract propagation rules. An important contrast point is that `CHR` interprets a rule as a template of which a copy with fresh variables is matched against the constraint store.

RuleMiner Rules. The work [AR00] describes the generation by an algorithm called "RuleMiner" of constraint propagation rules that in particular can have multiple constraints in its condition. The generated rules correspond to guard-free `CHR` propagation rules. The requirement that no new variables be introduced in the conclusion of a rule is not guaranteed by the generation algorithm. However, the algorithm takes special care to avoid introducing fresh variables. Indeed, each example presented in [AR00] fulfils the variable requirement.

Rule Generation Several criteria to discard a rule are used in the RuleMiner algorithm. The single most important is called *lhs-cover*. A rule $C'_L \to C'_R$ is lhs-covered by $C_L \to C_R$ if $C_L \subseteq C'_L$ and $C'_R \subseteq C_R$. This coincides with the condition of Corollary 8. The notion of lhs-cover is therefore a special case of general redundancy.

6 Experiments

We implemented in ECL^{i}PS^e [Ecl] the computation of minimal sets for membership rules and a class of RuleMiner rules. The results for some benchmark rule sets are listed in Tables 2, 3.

	$\texttt{and11}_M$	$\texttt{and11}_E$	$\texttt{and3}_M$	$\texttt{equ3}_M$	$\texttt{fula2}_E$	$\texttt{fork}_E$	$\texttt{fork}_M$
total	4656	153	18	26	52	12	24
(partially) redundant	4263 (2)	0 (6)	5 (0)	8 (0)	24 (0)	0 (9)	6 (6)
reduction ratio	9%	96%	70%	74%	65%	65%	60%

Table 2. Membership rules

Membership rules. For each constraint the set of minimal membership or equality rules (denoted by M/E) was computed by the rule generation algorithm of [AM01]. The table shows the total cardinality of the rule set, the number of fully and, in parentheses, partially redundant rules. The reduction ratio for the entire rule set shows how many atomic disequalities remain in the conclusions after redundancy removal.

The constraints are: the `and` logic gate for a number of logics (the numeric suffix states the domain size), the binary `fulladder` (arity 5), the equivalence relation for a three-valued logic, and the `fork` constraint from the Waltz language for the analysis of polyhedral scenes.

Computation times are unproblematic. The longest time required was 16 min for $\texttt{and11}_M$, much less than the corresponding rule generation time.

RuleMiner rules The authors of [AR00] kindly provided us with some RuleMiner-generated rule sets for the constraints `and`, `or`, `xor` in a 6-valued logic. Rules exist for propagation from single constraints, and propagation from pairs of constraints. In both cases additional equality constraints can occur in rule conditions. The conclusion of a rule consists of equalities and disequalities. Conclusions do not introduce fresh variables. The first three table rows describe

	`and`	`or`	`xor`	`andor`	`andxor`	`orxor`	`andor+`	`andxor+`	`orxor+`
total	19	19	28	138	207	199	176	254	246
partially redundant	7	7	1	83	82	77	135	192	184
reduction ratio	76%	76%	97%	62%	79%	79%	39%	46%	46%

Table 3. RuleMiner rules

the results for rule sets corresponding to the single constraints. The three center rows contain the results for rule sets for pairs. Finally, the last three rows correspond to the union of the rule sets for a pair of constraints and its respective individual constraints. This would be the appropriate use configuration.

The RuleMiner rule sets tested did not contain any fully redundant rules. This must probably be attributed to the additional redundancy criteria other than lhs-cover. The rule sets did contain a number of partial redundancies.

7 Outlook

Interesting issues to investigate further include the relation between two minimal sets. How many different minimal sets are there if we define equivalence classes on rules/rule sets such that, say, rule 10 and 11 of subsection 4.1 belong to one?

Rule generation (in particular of membership rules) could include a redundancy test as a part, not as a post-process. It is then useful to identify preconditions that prevent a rule from being redundant with respect to the partial rule set. One is that the body of the rule must occur in another rule.

A treatment of (a wider class of) `CHR` propagation rules as a kind of abstract propagation rule would be desirable. For instance, it appears possible to allow certain guards in rules. The partial orders for membership rules and abstract propagation rules could be combined into one, in which the inclusion test of a membership rule would correspond to a guard.

References

[AM01] Krzysztof R. Apt and Eric Monfroy. Constraint programming viewed as rule-based programming. *Theory and Practice of Logic Programming*, 2001.

[Apt99] Krzysztof R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1–2):179–210, June 1999.

[AR00] Slim Abdennadher and Christophe Rigotti. Automatic generation of propagation rules for finite domains. In *Principles and Practice of Constraint Programming (CP 2000), Singapore*, 2000.

[AR01] Slim Abdennadher and Christophe Rigotti. Towards inductive constraint solving. *Lecture Notes in Computer Science*, 2239, 2001.

[BKK+96] P. Borovansky, C. Kirchner, H. Kirchner, P. E. Moreau, and M. Vittek. ELAN: A logical framework based on computational systems. In *Workshop on Rewriting Logic*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996.

[Ecl] IC-Parc. *ECLiPSe*. `http://www.icparc.ic.ac.uk/eclipse/`.

[Frü98] Th. Frühwirth. Theory and practice of constraint handling rules. *Journal of Logic Programming*, pages 95–138, October 1998.

[RM00] C. Ringeissen and E. Monfroy. Generating propagation rules for finite domains via unification in finite algebras. In *New Trends in Constraints*, 2000.