

Unfolding for CHR programs

Maurizio Gabbrielli

*Department of Computer Science/Focus
Università di Bologna/INRIA
Mura Anteo Zamboni 7,
40127 Bologna, Italy
(e-mail: gabbri@cs.unibo.it)*

Maria Chiara Meo

*Dipartimento di Economia
Università di Chieti-Pescara
Viale Pindaro 42,
65127 Pescara, Italy
(e-mail: cmeo@unich.it)*

Paolo Tacchella

*Università di Bologna
Mura Anteo Zamboni 7,
40127 Bologna, Italy
(e-mail: Paolo.Tacchella@cs.unibo.it)*

Herbert Wiklicky

*Imperial College London
180 Queen's Gate
London SW7 2BZ, UK
(e-mail: herbert@doc.ic.ac.uk)*

submitted 25 October 2010; revised 12 September 2012; accepted 28 June 2013

Abstract

Program transformation is an appealing technique which allows to improve run-time efficiency, space-consumption, and more generally to optimize a given program. Essentially, it consists of a sequence of syntactic program manipulations which preserves some kind of semantic equivalence. Unfolding is one of the basic operations which is used by most program transformation systems and which consists in the replacement of a procedure call by its definition. While there is a large body of literature on transformation and unfolding of sequential programs, very few papers have addressed this issue for concurrent languages.

This paper defines an unfolding system for CHR programs. We define an unfolding rule, show its correctness and discuss some conditions which can be used to delete an unfolded rule while preserving the program meaning. We also prove that, under some suitable conditions, confluence and termination are preserved by the above transformation.

To appear in Theory and Practice of Logic Programming (TPLP).

KEYWORDS: CHR (Constraint Handling Rules), Program Transformation, Unfolding, Confluence, Termination.

1 Introduction

Constraint Handling Rules (CHR) (Frühwirth 1998; Frühwirth and Abdennadher 2003; Frühwirth 2009) is a concurrent, committed-choice language which was initially designed for writing constraint solvers and which is nowadays a general purpose language. A CHR program is a (finite) set of guarded rules, which allow to transform multisets of atomic formulas (constraints) into simpler ones.

There exists a very large body of literature on CHR, ranging from theoretical aspects to implementations and applications. However, only few papers, notably (Frühwirth and Holzbaaur 2003; Frühwirth 2005; Sneyers et al. 2005; Tacchella et al. 2007; Tacchella 2008; Sarna-Starosta and Schrijvers 2009), consider source to source transformation of CHR programs. This is not surprising, since program transformation is in general very difficult for (logic) concurrent languages and in case of CHR it is even more complicated, as we discuss later. Nevertheless, the study of this technique for concurrent languages and for CHR in particular, is important as it could lead to significant improvements in the run-time efficiency and space-consumption of programs.

Essentially, a source to source transformation consists of a sequence of syntactic program manipulations which preserves some kind of semantics. A basic manipulation is *unfolding*, which consists in the replacement of a procedure call by its definition. While this operation can be performed rather easily for sequential languages, and indeed in the field of logic programming it was first investigated by Tamaki and Sato more than twenty years ago (Tamaki and Sato 1984), when considering logic concurrent languages it becomes quite difficult to define reasonable conditions which ensure its correctness.

In this paper, we first define an unfolding rule for CHR programs and show that it preserves the semantics of the program in terms of qualified answers (Frühwirth 1998). Next, we provide a syntactic condition which allows one to replace in a program a rule by its unfolded version while preserving qualified answers. This condition preserves also termination, provided that one considers normal derivations. We also show that a more restricted condition ensures that confluence is preserved. Finally, we give a weaker condition for replacing a rule by its unfolded version: This condition allows to preserve the qualified answers for a specific class of programs (those which are normally terminating and confluent).

Even though the idea of the unfolding is straightforward, its technical development is complicated by the presence of guards, multiple heads, and matching substitution, as previously mentioned. In particular, it is not obvious to identify conditions which allow to replace the original rule by its unfolded version. Moreover, a further reason of complication comes from the fact that we consider as reference semantics the one defined in (Duck et al. 2004) and called ω_t , which avoids trivial non-termination by using a “token store” (or history). The token store idea was originally introduced by (Abdennadher 1997) but the shape of these tokens is different from that of those used in (Duck et al. 2004). Due to the presence of this token store, in order to define correctly the unfolding we have to slightly modify the syntax of CHR programs by adding to each rule a local token store. The re-

sulting programs are called annotated and we define their semantics by providing a (slightly) modified version of the semantics ω_t , which is proven to preserve the qualified answers.

The remainder of this paper is organized as follows. Section 2 introduces the CHR syntax while the operational semantics ω_t (Duck et al. 2004) and the modified one ω'_t are given in Section 3. Section 4 defines the unfolding rule (without replacement) and proves its correctness. Section 5 discuss the problems related to the replacement of a rule by its unfolded version and provides a correctness condition for such a replacement. In this section, we also prove that (normal) termination and confluence are preserved by the replacement which satisfies this condition. A further, weaker, condition ensuring the correctness of replacement for (normally) terminating and confluent programs is given in Section 6. Finally, Section 7 concludes by discussing some related works. Some of the proofs are deferred to the Appendix in order to improve the readability of the paper.

A preliminary version of this paper appeared in (Tacchella et al. 2007), some results were contained in the thesis (Tacchella 2008).

2 Preliminaries

In this section, we introduce the syntax of CHR and some notations and definitions we will need in the paper. For our purpose, a *constraint* is simply defined as an atom $p(t_1, \dots, t_n)$, where p is some predicate symbol of arity $n \geq 0$ and (t_1, \dots, t_n) is an n -tuple of terms. A *term* is (inductively) defined as a variable X , or as $f(t_1, \dots, t_n)$, where f is a function symbol of arity $n \geq 0$ and t_1, \dots, t_n are terms. \mathcal{T} is the set of all terms.

We use the following notation: let A be any syntactic object and let V be a set of variables. $\exists_V A$ denotes the existential closure of A w.r.t. the variables in V , while $\exists_{-V} A$ denotes the existential closure of A with the exception of the variables in V which remain unquantified. $Fv(A)$ denotes the free variables appearing in A .

We use “,” rather than \wedge to denote conjunction and we will often consider a conjunction of atomic constraints as a multiset of atomic constraints. We use $++$ for sequence concatenation, ϵ for empty sequence, \setminus for set difference operator and \uplus for multiset union. We shall sometimes treat multisets as sequences (or vice versa), in which case we nondeterministically choose an order for the objects in the multiset. We use the notation $p(s_1, \dots, s_n) = p(t_1, \dots, t_n)$ as a shorthand for the (conjunction of) constraints $s_1 = t_1, \dots, s_n = t_n$. Similarly if $S \equiv s_1, \dots, s_n$ and $T \equiv t_1, \dots, t_n$ are sequences of equal length then $S = T$ is a shorthand for $s_1 = t_1, \dots, s_n = t_n$.

A substitution is a mapping $\vartheta : V \rightarrow \mathcal{T}$ such that the set $dom(\vartheta) = \{X \mid \vartheta(X) \neq X\}$ (domain of ϑ) is finite; ε is the empty substitution: $dom(\varepsilon) = \emptyset$.

The composition $\vartheta\sigma$ of the substitutions ϑ and σ is defined as the functional composition. A substitution ϑ is idempotent if $\vartheta\vartheta = \vartheta$. A renaming is a (nonidempotent) substitution ρ for which there exists the inverse ρ^{-1} such that $\rho\rho^{-1} = \rho^{-1}\rho = \varepsilon$.

We restrict our attention to idempotent substitutions, unless explicitly stated otherwise.

Constraints can be divided into either *user-defined* (or CHR) constraints or *built-in* constraints on some constraint domain \mathcal{D} . The built-in constraints are handled by an existing solver and we assume given a (first order) theory \mathcal{CT} which describes their meaning. We assume also that the built-in constraints contain the predicate $=$ which is described, as usual, by the Clark Equality Theory (Lloyd 1984) and the values **true** and **false** with their obvious meaning.

We use c, d to denote built-in constraints, h, k, f, s, p, q to denote CHR constraints, and a, b, g to denote both built-in and user-defined constraints (we will call these generically constraints). The capital versions will be used to denote multisets (or sequences) of constraints.

2.1 CHR syntax

As shown by the following definition (Frühwirth 1998), a *CHR program* consists of a set of rules which can be divided into three types: *simplification*, *propagation*, and *simpagation* rules. The first kind of rules is used to rewrite CHR constraints into simpler ones, while second kind allows to add new redundant constraints which may cause further simplification. Simpagation rules allow to represent both simplification and propagation rules.

Definition 1 (CHR SYNTAX)

A CHR program is a finite set of CHR rules. There are three kinds of CHR rules: A **simplification** rule has the form:

$$r@H \Leftrightarrow C \mid B$$

A **propagation** rule has the form:

$$r@H \Rightarrow C \mid B$$

A **simpagation** rule has the form:

$$r@H_1 \setminus H_2 \Leftrightarrow C \mid B,$$

where r is a unique identifier of a rule, H , H_1 and H_2 are sequences of user-defined constraints, with H and $H_1 \ ++ \ H_2$ different from the empty sequence, C is a possibly empty conjunction of built-in constraints, and B is a possibly empty sequence of (built-in and user-defined) constraints. H (or $H_1 \setminus H_2$) is called *head*, C is called *guard* and B is called *body* of the rule.

A *simpagation* rule can simulate both simplification and propagation rule by considering, respectively, either H_1 or H_2 empty. In the following, we will consider in the formal treatment only simpagation rules.

2.2 CHR Annotated syntax

When considering unfolding we need to consider a slightly different syntax, where rule identifiers are not necessarily unique, each atom in the body is associated with an identifier, that is unique in the rule, and where each rule is associated with a local token store T . More precisely, we define an identified CHR constraint (or identified atom) $h\#i$ as a CHR constraint h associated with an integer i which allows to distinguish different copies of the same constraint.

Moreover, let us define a token as an object of the form $r@i_1, \dots, i_l$, where r is the name of a rule and i_1, \dots, i_l is a sequence of distinct identifiers. A token store (or history) is a set of tokens.

Definition 2 (CHR ANNOTATED SYNTAX)

An *annotated* rule has then the form:

$$r@H_1 \setminus H_2 \Leftrightarrow C \mid B; T$$

where r is an identifier, H_1 and H_2 are sequences of user-defined constraints with $H_1 ++ H_2$ different from the empty sequence, C is a possibly empty conjunction of built-in constraints, B is a possibly empty sequence of built-in and identified CHR constraints such that different (occurrences of) CHR constraints have different identifiers, and T is a token store. $H_1 \setminus H_2$ is called *head*, C is called *guard*, B is called *body* and T is called *local token store* of the annotated rule. An annotated CHR program is a finite set of annotated CHR rules.

We will also use the following two functions: $chr(h\#i) =_{def} h$ and the overloaded function $id(h\#i) =_{def} i$, (and $id(r@i_1, \dots, i_l) =_{def} \{i_1, \dots, i_l\}$), extended to sets and sequences of identified CHR constraints (or tokens) in the obvious way. An (identified) CHR *goal* is a multi-set of both (identified) user-defined and built-in constraints. *Goals* is the set of all (possibly identified) goals.

Intuitively, identifiers are used to distinguish different occurrences of the same atom in a rule or in a goal. The identified atoms can be obtained by using a suitable function which associates a (unique) integer to each atom. More precisely, let B be a goal which contains m CHR-constraints. We assume that the function $I(B)$ identifies each CHR constraint in B by associating to it a unique integer in $[1, m]$ according to the lexicographic order.

The token store allows one to memorize some tokens, where each token describes which propagation rule has been used for reducing which identified atoms. As we discuss in the next section, the use of this information was originally proposed in (Abdennadher 1997) and then further elaborated in the semantics defined in (Duck et al. 2004) in order to avoid trivial non-termination arising from the repeated application of the same propagation rule to the same constraints. Here, we simply incorporate this information in the syntax, since we will need to manipulate it in our unfolding rule.

Given a CHR program P , by using the function $I(B)$ and an initially empty local token store we can construct its annotated version as the next definition explains.

Definition 3

Let P be a CHR program. Then its annotated version is defined as follows:

$$Ann(P) = \left\{ \begin{array}{l} r@H_1 \setminus H_2 \Leftrightarrow C \mid I(B); \emptyset \text{ such that} \\ r@H_1 \setminus H_2 \Leftrightarrow C \mid B \in P \end{array} \right\}.$$

Notation

In the following examples, given a (possibly annotated) rule

$$r@H_1 \setminus H_2 \Leftrightarrow C \mid B(;T),$$

we write it as

$$r@H_2 \Leftrightarrow C \mid B(;T),$$

if H_1 is empty and we write it as

$$r@H_1 \Rightarrow C \mid B(;T),$$

if H_2 is empty. That is, we maintain also the notation previously introduced for simplification and propagation rules. Moreover, if $C = \mathbf{true}$, then $\mathbf{true} \mid$ is omitted and if in an annotated rule the token store is empty we simply omit it. Sometimes, in order to simplify the notation, if in an annotated program P there are no annotated propagation rules, then we write P by using the standard syntax.

Finally, we will use cl, cl', \dots to denote (possibly annotated) rules and cl_r, cl'_r, \dots to denote (possibly annotated) rules with identifier r .

Example 1

The following CHR program, given a forest of finite trees (defined in terms of the predicates `root` and `edge`, with the obvious meaning), is able to recognize if two nodes belong to the same tree and if so returns the root.

The program P consists of the following five rules

$$\begin{array}{l} r_1@root(V), same(X, Y) \Rightarrow X = Y, X = V \mid success(V) \\ r_2@root(V), same(X, Y) \Leftrightarrow X \neq Y \mid root(V), same(V, X), path(V, Y) \\ r_3@path(I, J) \Rightarrow I = J \mid \mathbf{true} \\ r_4@edge(U, Z) \setminus path(I, J) \Leftrightarrow J = Z \mid path(I, U) \\ r_5@root(V) \setminus path(I, J) \Leftrightarrow V = J, V \neq I \mid \mathbf{false} \end{array}$$

Then its annotated version $Ann(P)$ is defined as follows:

$$\begin{array}{l} r_1@root(V), same(X, Y) \Rightarrow X = Y, X = V \mid success(V)\#1; \emptyset \\ r_2@root(V), same(X, Y) \Leftrightarrow X \neq Y \mid root(V)\#1, same(V, X)\#2, path(V, Y)\#3; \emptyset \\ r_3@path(I, J) \Rightarrow I = J \mid \mathbf{true}; \emptyset \\ r_4@edge(U, Z) \setminus path(I, J) \Leftrightarrow J = Z \mid path(I, U)\#1; \emptyset \\ r_5@root(V) \setminus path(I, J) \Leftrightarrow V = J, V \neq I \mid \mathbf{false}; \emptyset \end{array}$$

3 CHR operational semantics

This section first introduces the reference semantics ω_t (Duck et al. 2004). For the sake of simplicity, we omit indexing the relation with the name of the program.

Table 1. The transition system T_{ω_t} for the ω_t semantics

Solve	$\frac{c \text{ is a built-in constraint}}{\langle \{c\} \uplus G, S, C, T \rangle_n \longrightarrow_{\omega_t} \langle G, S, C \wedge c, T \rangle_n}$
Introduce	$\frac{h \text{ is a user-defined constraint}}{\langle \{h\} \uplus G, S, C, T \rangle_n \longrightarrow_{\omega_t} \langle G, \{h\#n\} \uplus S, C, T \rangle_{n+1}}$
Apply	$\frac{\begin{array}{l} r@H'_1 \setminus H'_2 \Leftrightarrow D \mid B \in P \\ \mathcal{CT} \models C \rightarrow \exists_r((chr(H_1, H_2) = (H'_1, H'_2)) \wedge D) \end{array}}{\begin{array}{l} \langle G, H_1 \uplus H_2 \uplus S, C, T \rangle_n \longrightarrow_{\omega_t} \\ \langle B \uplus G, H_1 \uplus S, (chr(H_1, H_2) = (H'_1, H'_2)) \wedge D \wedge C, T' \rangle_n \end{array}}$ <p style="margin-top: 5px;">where $r@id(H_1, H_2) \notin T$ and $T' = T \cup \{r@id(H_1, H_2)\}$</p>

Next, we define a slightly different operational semantics, called ω'_t , which considers annotated programs and which will be used to prove the correctness of our unfolding rules (via some form of equivalence between ω'_t and ω_t).

In the following, given a (possibly annotated) rule $cl_r = r@H_1 \setminus H_2 \Leftrightarrow C \mid B(;T)$, we denote by \exists_{cl_r} the existential quantification $\exists_{Fv(H_1, H_2, C, B)}$. By an abuse of notation, when it is clear from the context, we will write \exists_r instead of \exists_{cl_r} .

3.1 The semantics ω_t

We describe the operational semantics ω_t , introduced in (Duck et al. 2004), by using a transition system

$$T_{\omega_t} = (Conf_t, \longrightarrow_{\omega_t}).$$

Configurations in $Conf_t$ are tuples of the form $\langle G, S, C, T \rangle_n$ where G , the *goal store* is a multiset of constraints. The *CHR constraint store* S is a set of identified CHR constraints. The *built-in constraint store* C is a conjunction of built-in constraints. The *propagation history* T is a token store and n is an integer. Throughout this paper, we use the symbols $\sigma, \sigma', \sigma_i, \dots$ to represent configurations in $Conf_t$.

The *goal store* (G) contains all constraints to be executed. The *CHR constraint store* (S) is the set¹ of identified CHR constraints that can be matched with the head of the rules in the program P . The *built-in constraint store* (C) contains any built-in constraint that has been passed to the built-in constraint solver. Since we will usually have no information about the internal representation of C , we treat it as a conjunction of constraints. The *propagation history* (T) describes which rule has been used for reducing which identified atoms. Finally, the *counter* n represents the next free integer which can be used to number a CHR constraint.

¹ Note that sometimes we treat S as a multiset. This is the case, for example, of the transition rules, where considering S as a multiset simplifies the notation.

Given a goal G , the *initial configuration* has the form

$$\langle G, \emptyset, \mathbf{true}, \emptyset \rangle_1.$$

A *final configuration* has either the form $\langle G', S, \mathbf{false}, T \rangle_n$, when it is *failed*, or it has the form $\langle \emptyset, S, C, T \rangle_n$ (with $\mathcal{CT} \models C \not\rightarrow \mathbf{false}$) when it represents a successful termination (since there are no more applicable rules).

The relation \rightarrow_{ω_t} (of the transition system T_{ω_t}) is defined by the rules in Table 1: the **Solve** rule moves a built-in constraint from the goal store to the built-in constraint store; the **Introduce** rule identifies and moves a CHR (or user-defined) constraint from the goal store to the CHR constraint store; the **Apply** rule chooses a program rule cl and fires it, provided that the following conditions are satisfied: there exists a matching between the constraints in the CHR store and the ones in the head of cl ; the guard of cl is entailed by the built-in constraint store (taking into account also the matching mentioned before); the token that would be added by **Apply** to the token store is not already present. After the application of cl , the constraints which match with the right hand side of the head of cl are deleted from the CHR constraint store, the body of cl is added to the goal store and the guard of cl , together with the equality representing the matching, is added to the built-in constraint store. The **Apply** rule assumes that all the variables appearing in a program clause are renamed with fresh ones in order to avoid variable names clashes.

From the rules, it is clear that when not considering tokens (as in the original semantics of (Frühwirth 1998)) if a propagation rule can be applied once then it can be applied infinitely many times, thus producing an infinite computation (no fairness assumptions are made here). Such a trivial non-termination is avoided by tokens, since they ensure that if a propagation rule is used to reduce a sequence of constraints then the same rule has not been used before on the same sequence of constraints.

3.2 The modified semantics ω'_t

We now define the semantics ω'_t which considers annotated rules. This semantics differs from ω_t in two aspects.

First, in ω'_t the goal store and the CHR store are fused in a unique generic *store*, where CHR constraints are immediately labeled. As a consequence, we do not need the **Introduce** rule anymore and every CHR constraint in the body of an applied rule is immediately utilizable for rewriting.

The second difference concerns the shape of the rules. In fact, each annotated rule cl has a local token store (which can be empty) that is associated with it and which is used to keep track of the propagation rules that are used to unfold the body of cl . Note also that here, differently from the case of the propagation history in ω_t , the token store associated with a computation can be updated by adding multiple tokens at once (because an unfolded rule with many tokens in its local token store has been used).

In order to define ω'_t formally, we need a function *inst* which updates the formal

identifiers of a rule to the actual computation ones. Such a function is defined as follows.

Definition 4

Let $Token$ be the set of all possible token sets and let \mathbb{N} be the set of natural numbers. We denote by $inst : Goals \times Token \times \mathbb{N} \rightarrow Goals \times Token \times \mathbb{N}$ the function such that $inst(B, T, n) = (B', T', m)$, where

- B is an identified CHR goal,
- (B', T') is obtained from (B, T) by incrementing each identifier in (B, T) with n and
- m is the greatest identifier in (B', T') .

We describe now the operational semantics ω'_t for annotated CHR programs by using, as usual, a transition system

$$T_{\omega'_t} = (Conf'_t, \longrightarrow_{\omega'_t}).$$

Configurations in $Conf'_t$ are tuples of the form $\langle S, C, T \rangle_n$ with the following meaning. S is the set² of identified CHR constraints that can be matched with rules in the program P and built-in constraints. The built-in constraint store C is a conjunction of built-in constraints and T is a set of tokens, while the counter n represents the last integer which was used to number the CHR constraints in S .

Given a goal G , the *initial configuration* has the form

$$\langle I(G), \mathbf{true}, \emptyset \rangle_m,$$

where m is the number of CHR constraints in G and I is the function which associates the identifiers with the CHR constraints in G . A *failed configuration* has the form $\langle S, \mathbf{false}, T \rangle_n$.

A *final configuration* either is failed or it has the form $\langle S, C, T \rangle_n$ (with $CT \models C \not\vdash \mathbf{false}$) when it represents a successful termination, since there are no more applicable rules.

The relation $\longrightarrow_{\omega'_t}$ (of the transition system $T_{\omega'_t}$) is defined by the rules in Table 2 which have the following explanation:

Solve' moves a built-in constraint from the store to the built-in constraint store;
Apply' fires a rule cl of the form $r@H'_1 \setminus H'_2 \Leftrightarrow D \mid B; T_r$ provided that the following conditions are satisfied: there exists a matching between the constraints in the store and the ones in the head of cl ; the guard of cl is entailed by the built-in constraint store (taking into account also the matching mentioned before); $r@id(H_1, H_2) \notin T$. These conditions are equal to those already seen for **Apply**. Moreover, analogously to the **Apply** transition step, $chr(H_1, H_2) = (H'_1, H'_2)$ together with D are added to the built-in constraint store. However, in this case, when the rule cl is fired, H_2 is replaced by B and the local store T_r is added to T (with $r@id(H_1, H_2)$), where each identifier is suitably incremented by the $inst$

² Also in this case, sometimes we treat S as a multiset. See the previous footnote.

Table 2. The transition system $T_{\omega'_t}$ for the ω'_t semantics

Solve'	$\frac{c \text{ is a built-in constraint}}{\langle \{c\} \uplus G, C, T \rangle_n \longrightarrow_{\omega'_t} \langle G, c \wedge C, T \rangle_n}$
Apply'	$\frac{\begin{array}{l} r @ H'_1 \setminus H'_2 \Leftrightarrow D \mid B; T_r \in P, \\ CT \models C \rightarrow \exists_r ((chr(H_1, H_2) = (H'_1, H'_2)) \wedge D) \end{array}}{\langle H_1 \uplus H_2 \uplus G, C, T \rangle_n \longrightarrow_{\omega'_t} \langle B' \uplus H_1 \uplus G, (chr(H_1, H_2) = (H'_1, H'_2)) \wedge D \wedge C, T' \rangle_m}$

where $(B', T'_r, m) = inst(B, T_r, n)$, $r @ id(H_1, H_2) \notin T$ and $T' = T \cup \{r @ id(H_1, H_2)\} \cup T'_r$.

function. Finally, the subscript n is replaced by m , that is the greatest number used during the computation step.

As for the **Apply** rule, the **Apply'** rule assumes that all the variables appearing in a program clause are renamed with fresh ones in order to avoid variable names clashes.

The following example shows a derivation obtained by the new transition system.

Example 2

Given the goal $root(a), same(b, c), edge(a, b), edge(a, d), edge(d, c)$ in the following program P' ,

```

r1@root(V), same(X, Y) ⇒ X = Y, X = V | success(V)#1; ∅
r2@root(V), same(X, Y) ⇔ X ≠ Y | root(V)#1, same(V, X)#2, path(V, Y)#3; ∅
r2@root(V), same(X, Y) ⇔ X ≠ Y, V = X | root(V)#1, same(V, X)#2, path(V, Y)#3,
    success(I)#4, V = I, V = J, X = L; {r1@1, 2}
r2@root(V), same(X, Y) ⇔ X ≠ Y, V ≠ X | path(V, Y)#3, root(I)#4, same(J, L)#5,
    I = V, J = V, L = X; ∅
r2@root(V), same(X, Y) ⇔ X ≠ Y, V = Y | root(V)#1, same(V, X)#2,
    path(V, Y)#3, V = I, Y = J; {r3@3}

r3@path(I, J) ⇒ I = J | true; ∅
r4@edge(U, Z) \ path(I, J) ⇔ J = Z | path(I, U)#1; ∅
r4@edge(U, Z) \ path(I, J) ⇔ J = Z, I = U | path(I, U)#1, I = X, U = Y; {r3@1}
r5@root(V) \ path(I, J) ⇔ V = J, V ≠ I | false; ∅

```

we obtain the following derivation

$$\langle \langle \mathbf{root(a)\#1, same(b, c)\#2, edge(a, b)\#3, edge(a, c)\#4, edge(c, d)\#5}, true, \emptyset \rangle_5 \longrightarrow_{\omega'_t} \langle \langle \text{path}(V_1, Y_1)\#6, \text{root}(I_1)\#7, \text{same}(J_1, L_1)\#8, I_1 = V_1, J_1 = V_1, L_1 = X_1, \text{edge}(a, b)\#3, \text{edge}(a, d)\#4, \text{edge}(d, c)\#5}, (a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \{r_2 @ 1, 2\} \rangle_8 \longrightarrow_{\omega'_t}^* \langle \langle \text{path}(V_1, Y_1)\#6, \mathbf{root(I_1)\#7}, \mathbf{same(J_1, L_1)\#8}, \text{edge}(a, b)\#3, \text{edge}(a, c)\#4, \text{edge}(c, d)\#5}, (I_1 = V_1, J_1 = V_1, L_1 = X_1, a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \{r_2 @ 1, 2\} \rangle_8 \longrightarrow_{\omega'_t}$$

$$\begin{aligned}
& \langle (\text{root}(V_2)\#9, \text{same}(V_2, X_2)\#10, \text{path}(V_2, Y_2)\#11, \text{success}(I_2)\#12, \\
& V_2 = I_2, V_2 = J_2, X_2 = L_2, \text{path}(V_1, Y_1)\#6, \text{edge}(a, b)\#3, \text{edge}(a, c)\#4, \text{edge}(c, d)\#5), \\
& (V_2 = I_1, X_2 = J_1, Y_2 = L_1, X_2 \neq Y_2, V_2 = X_2, I_1 = V_1, J_1 = V_1, L_1 = X_1, \\
& a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \{r_2@1, 2, r_2@7, 8, r_1@9, 10\}\rangle_{12} \xrightarrow{\omega'_t} \\
& \langle (\text{root}(V_2)\#9, \text{same}(V_2, X_2)\#10, \mathbf{path}(V_2, Y_2)\#11, \text{success}(I_2)\#12, \\
& \mathbf{path}(V_1, Y_1)\#6, \mathbf{edge}(a, b)\#3, \text{edge}(a, c)\#4, \text{edge}(c, d)\#5), \\
& (V_2 = I_2, V_2 = J_2, X_2 = L_2, V_2 = I_1, X_2 = J_1, Y_2 = L_1, X_2 \neq Y_2, V_2 = X_2, I_1 = V_1, J_1 = V_1, \\
& L_1 = X_1, a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \{r_2@1, 2, r_2@7, 8, r_1@9, 10\}\rangle_{12} \xrightarrow{\omega'_t} \\
& \langle (\text{path}(I_3, U_3)\#13, I_3 = X_3, U_3 = Y_3, \text{root}(V_2)\#9, \text{same}(V_2, X_2)\#10, \\
& \text{success}(I_2)\#12, \text{path}(V_1, Y_1)\#6, \text{edge}(a, b)\#3, \text{edge}(a, c)\#4, \text{edge}(c, d)\#5), \\
& (a = U_3, b = Z_3, a = I_3, b = J_3, J_3 = Z_3, I_3 = U_3, V_2 = I_2, V_2 = J_2, X_2 = L_2, V_2 = I_1, \\
& X_2 = J_1, Y_2 = L_1, X_2 \neq Y_2, V_2 = X_2, I_1 = V_1, J_1 = V_1, L_1 = X_1, a = V_1, b = X_1, \\
& c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \{r_2@1, 2, r_2@7, 8, r_1@9, 10, r_4@11, 3, r_3@13\}\rangle_{13} \xrightarrow{\omega'_t} \\
& \langle (\text{path}(I_3, U_3)\#13, \text{root}(V_2)\#9, \text{same}(V_2, X_2)\#10, \text{success}(I_2)\#12, \\
& \mathbf{path}(V_1, Y_1)\#6, \text{edge}(a, b)\#3, \mathbf{edge}(a, c)\#4, \text{edge}(c, d)\#5), \\
& (I_3 = X_3, U_3 = Y_3, a = U_3, b = Z_3, a = I_3, b = J_3, J_3 = Z_3, I_3 = U_3, V_2 = I_2, \\
& V_2 = J_2, X_2 = L_2, V_2 = I_1, X_2 = J_1, Y_2 = L_1, X_2 \neq Y_2, V_2 = X_2, I_1 = V_1, J_1 = V_1, \\
& L_1 = X_1, a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \\
& \{r_2@1, 2, r_2@7, 8, r_1@9, 10, r_4@11, 3, r_3@13\}\rangle_{13} \xrightarrow{\omega'_t} \\
& \langle (\text{path}(I_4, U_4)\#14, I_4 = X_4, U_4 = Y_4, \text{path}(I_3, U_3)\#13, \text{root}(V_2)\#9, \\
& \text{same}(V_2, X_2)\#10, \text{success}(I_2)\#12, \text{edge}(a, b)\#3, \text{edge}(a, c)\#4, \text{edge}(c, d)\#5), \\
& (a = U_4, c = Z_4, V_1 = I_4, Y_1 = J_4, J_4 = Z_4, I_4 = U_4, I_3 = X_3, U_3 = Y_3, a = U_3, b = Z_3, \\
& a = I_3, b = J_3, J_3 = Z_3, I_3 = U_3, V_2 = I_2, V_2 = J_2, X_2 = L_2, V_2 = I_1, X_2 = J_1, Y_2 = L_1, \\
& X_2 \neq Y_2, V_2 = X_2, I_1 = V_1, J_1 = V_1, L_1 = X_1, a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \\
& \{r_2@1, 2, r_2@7, 8, r_1@9, 10, r_4@11, 3, r_3@13, r_4@4, 6, r_3@14\}\rangle_{14} \xrightarrow{\omega'_t} \\
& \langle (\text{path}(I_4, U_4)\#14, \text{path}(I_3, U_3)\#13, \text{root}(V_2)\#9, \text{same}(V_2, X_2)\#10, \\
& \text{success}(I_2)\#12, \text{edge}(a, b)\#3, \text{edge}(a, c)\#4, \text{edge}(c, d)\#5), \\
& (I_4 = X_4, U_4 = Y_4, a = U_4, c = Z_4, V_1 = I_4, Y_1 = J_4, J_4 = Z_4, I_4 = U_4, I_3 = X_3, \\
& U_3 = Y_3, a = U_3, b = Z_3, a = I_3, b = J_3, J_3 = Z_3, I_3 = U_3, V_2 = I_2, V_2 = J_2, \\
& X_2 = L_2, V_2 = I_1, X_2 = J_1, Y_2 = L_1, X_2 \neq Y_2, V_2 = X_2, I_1 = V_1, J_1 = V_1, L_1 = X_1, \\
& a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \\
& \{r_2@1, 2, r_2@7, 8, r_1@9, 10, r_4@11, 3, r_3@13, r_4@4, 6, r_3@14\}\rangle_{14} \not\xrightarrow{\omega'_t}
\end{aligned}$$

From the previous transition systems we can obtain a notion of observable property of CHR computations that will be used in order to prove the correctness of our unfolding rule. The notion of "observable property" usually identifies the relevant property that one is interested in observing as the result of a computation. In our case, we use the notion of qualified answer, originally introduced in (Frühwirth 1998): Intuitively this is the constraint obtained as the result of a non-failed computation, including both built-in constraints and CHR constraints which have not been "solved" (i.e. transformed by rule applications into built-in constraints). Formally qualified answer are defined as follows.

Definition 5 (QUALIFIED ANSWERS)

Let P be a CHR program and let G be a goal. The set $\mathcal{QA}_P(G)$ of qualified answers

for the query G in the program P is defined as follows:

$$\mathcal{QA}_P(G) = \{\exists_{-Fv(G)}(chr(K) \wedge D) \mid \mathcal{CT} \not\models D \leftrightarrow \mathbf{false} \text{ and} \\ \langle G, \emptyset, \mathbf{true}, \emptyset \rangle_1 \rightarrow_{\omega_t}^* \langle \emptyset, K, D, T \rangle_n \not\rightarrow_{\omega_t}\}.$$

Analogously, we can define the qualified answer of an annotated program.

Definition 6 (QUALIFIED ANSWERS FOR ANNOTATED PROGRAMS)

Let P be an annotated CHR program and let G be a goal. The set $\mathcal{QA}'_P(G)$ of qualified answers for the query G in the annotated program P is defined as follows:

$$\mathcal{QA}'_P(G) = \{\exists_{-Fv(G)}(chr(K) \wedge D) \mid \mathcal{CT} \not\models D \leftrightarrow \mathbf{false} \text{ and} \\ \langle I(G), \mathbf{true}, \emptyset \rangle_m \rightarrow_{\omega'_t}^* \langle K, D, T \rangle_n \not\rightarrow_{\omega'_t}\}.$$

The previous two notions of qualified answers are equivalent, as shown by the proof (in the Appendix) of the following proposition. This fact will be used to prove the correctness of the unfolding.

Proposition 1

Let P and $Ann(P)$ be respectively a CHR program and its annotated version. Then, for every goal G ,

$$\mathcal{QA}_P(G) = \mathcal{QA}'_{Ann(P)}(G)$$

holds.

4 The unfolding rule

In this section, we define the *unfold operation* for CHR simpagation rules. As a particular case, we obtain also unfolding for simplification and propagation rules, as these can be seen as particular cases of the former.

The unfolding allows to replace a conjunction S of constraints (which can be seen as a procedure call) in the body of a rule cl_r by the body of a rule cl_v , provided that the head of cl_v matches with S (when considering also the instantiations provided by the built-in constraints in the guard and in the body of the rule cl_r). More precisely, assume that the built-in constraints in the guard and in the body of the rule cl_r imply that the head H of cl_v , instantiated by a substitution θ , matches with the conjunction S in the body of cl_r . Then, the unfolded rule is obtained from cl_r by performing the following steps: 1) the new guard in the unfolded rule is the conjunction of the guard of cl_r with the guard of cl_v , the latter instantiated by θ and without those constraints that are entailed by the built-in constraints which are in cl_r ; 2) the body of cl_v and the equality $H = S$ are added to the body of cl_r ; 3) the conjunction of constraints S can be removed, partially removed or left in the body of the unfolded rule, depending on the fact that cl_v is a simplification, a simpagation or a propagation rule, respectively; 4) as for the local token store T_r associated with every rule cl_r , this is updated consistently during the unfolding operations in order to avoid that a propagation rule is used twice to unfold the same sequence of constraints.

Before giving the formal definition of the unfolding rule, we illustrate the above steps by means of the following example.

Example 3

Consider the following program P , similar to that one given in (Schrijvers and Sulzmann 2008), which describes the rules for updating a bank account and for performing the money transfer. We write the program by using the standard syntax, namely without using the local token store and the identifiers in the body of rules, since there are no annotated propagation rules. The program P consists of the following three rules

$$\begin{aligned} r_1 @ & b(\text{Acc1}, \text{Bal1}), b(\text{Acc2}, \text{Bal2}), t(\text{Acc1}, \text{Acc2}, \text{Amount}) \Leftrightarrow \text{Acc1} \neq \text{Acc2} \mid \\ & b(\text{Acc1}, \text{Bal1}), b(\text{Acc2}, \text{Bal2}), w(\text{Acc1}, \text{Amount}), d(\text{Acc2}, \text{Am}) \\ r_2 @ & b(\text{Acc}, \text{Bal}), d(\text{Acc}, \text{Am}) \Leftrightarrow b(\text{Acc}, \text{B}), \text{B} = \text{Bal} + \text{Am} \\ r_3 @ & b(\text{Acc}', \text{Bal}'), w(\text{Acc}', \text{Am}') \Leftrightarrow \text{Bal}' > \text{Amount}' \mid b(\text{Acc}', \text{B}'), \text{B}' = \text{Bal}' - \text{Am}' \end{aligned}$$

where the three rules identified by r_1, r_2 , and r_3 are called cl_{r_1}, cl_{r_2} , and cl_{r_3} , respectively. The predicate names are abbreviations: b for balance, d for deposit, w for withdraw and t for transfer.

Now, we unfold the rule cl_{r_1} by using the rule cl_{r_2} and we obtain the new clause cl'_{r_1} :

$$\begin{aligned} r_1 @ & b(\text{Acc1}, \text{Bal1}), b(\text{Acc2}, \text{Bal2}), t(\text{Acc1}, \text{Acc2}, \text{Amount}) \Leftrightarrow \text{Acc1} \neq \text{Acc2} \mid \\ & b(\text{Acc1}, \text{Bal1}), w(\text{Acc1}, \text{Amount}), b(\text{Acc}, \text{B}), \\ & \text{B} = \text{Bal} + \text{Am}, \text{Acc2} = \text{Acc}, \text{Bal2} = \text{Bal}, \text{Amount} = \text{Am}. \end{aligned}$$

Next, we unfold the rule cl'_{r_1} by using the rule cl_{r_3} and we can obtain the new clause cl''_{r_3}

$$\begin{aligned} r_3 @ & b(\text{Acc1}, \text{Bal1}), b(\text{Acc2}, \text{Bal2}), t(\text{Acc1}, \text{Acc2}, \text{Amount}) \Leftrightarrow \\ & \text{Acc1} \neq \text{Acc2}, \text{Bal1} > \text{Amount} \mid b(\text{Acc}, \text{B}), \text{B} = \text{Bal} + \text{Am}, \text{Acc2} = \text{Acc}, \text{Bal2} = \text{Bal}, \\ & \text{Amount} = \text{Am}, b(\text{Acc}', \text{B}'), \text{B}' = \text{Bal}' - \text{Am}', \\ & \text{Acc1} = \text{Acc}', \text{Bal1} = \text{Bal}', \text{Amount} = \text{Am}'. \end{aligned}$$

Before formally defining the unfolding, we need to define a function which removes the useless tokens from the token store.

Definition 7

Let B be an identified goal and let T be a token set,

$$\text{clean} : \text{Goals} \times \text{Token} \rightarrow \text{Token},$$

is defined as follows: $\text{clean}(B, T)$ deletes from T all the tokens for which at least one identifier is not present in the identified goal B . More formally

$$\text{clean}(B, T) = \{t \in T \mid t = r @ i_1, \dots, i_k \text{ and } i_j \in \text{id}(B), \text{ for each } j \in [1, k]\}.$$

Definition 8 (UNFOLD)

Let P be an annotated CHR program and let $cl_r, cl_v \in P$ be the two following annotated rules

$$\begin{aligned} r @ H_1 \setminus H_2 & \Leftrightarrow D \mid K, S_1, S_2, C; T \text{ and} \\ v @ H'_1 \setminus H'_2 & \Leftrightarrow D' \mid B; T' \end{aligned}$$

respectively, where C is the conjunction of all the built-in constraints in the body of cl_r . Let θ be a substitution such that $\text{dom}(\theta) \subseteq \text{Fv}(H'_1, H'_2)$ and $\mathcal{CT} \models (C \wedge D) \rightarrow \text{chr}(S_1, S_2) = (H'_1, H'_2)\theta$. Furthermore let m be the greatest identifier which appears in the rule cl_r and let $(B_1, T_1, m_1) = \text{inst}(B, T', m)$. Then, the *unfolded* rule is:

$$r @ H_1 \setminus H_2 \Leftrightarrow D, (D''\theta) \mid K, S_1, B_1, C, \text{chr}(S_1, S_2) = (H'_1, H'_2); T''$$

where $v@id(S_1, S_2) \notin T$, $V = \{d \in D' \mid \mathcal{CT} \models C \wedge D \rightarrow d\theta\}$, $D'' = D' \setminus V$, $Fv(D''\theta) \cap Fv(H'_1, H'_2)\theta \subseteq Fv(H_1, H_2)$, the constraint $(D, (D''\theta))$ is satisfiable and

- if $H'_2 = \epsilon$ then $T'' = T \cup T_1 \cup \{v@id(S_1)\}$
- if $H'_2 \neq \epsilon$ then $T'' = \text{clean}((K, S_1), T) \cup T_1$.

Note that $V \subseteq D'$ is the greatest set of built-in constraints such that $\mathcal{CT} \models C \wedge D \rightarrow d\theta$ for each $d \in V$. Moreover, as shown in the following, all the results in the paper are independent from the choice of the substitution θ which satisfies the conditions of Definition 8. Finally, we use the function *inst* (Definition 4) in order to increment the value of the identifiers associated with atoms in the unfolded rule. This allows us to distinguish the new identifiers introduced in the unfolded rule from the old ones. Note also that the condition on the token store is needed to obtain a correct rule. Consider for example a ground annotated program

$$P = \left\{ \begin{array}{l} r_1@h \Leftrightarrow k\#1 \\ r_2@k \Rightarrow s\#1 \\ r_3@s, s \Leftrightarrow q\#1 \end{array} \right\}$$

where the three rules identified by r_1, r_2 , and r_3 are called cl_{r_1}, cl_{r_2} , and cl_{r_3} , respectively³. Let h be the start goal. In this case, the unfolding could change the semantics if the token store was not used. In fact, according to the semantics proposed in Table 1 or 2, we have that the goal h has only the qualified answer (k, s) . On the other hand, considering an unfolding without the update of the token store, one would have $r_1@h \Leftrightarrow k\#1 \xrightarrow{\text{unfold using } cl_{r_2}} r_1@h \Leftrightarrow k\#1, s\#2 \xrightarrow{\text{unfold using } cl_{r_2}} r_1@h \Leftrightarrow k\#1, s\#2, s\#3 \xrightarrow{\text{unfold using } cl_{r_3}} r_1@h \Leftrightarrow k\#1, q\#4$. So, starting from the constraint h we could obtain the qualified answer (k, q) , that is not possible in the original program (the rule obtained after the wrongly applied unfolding rule is underlined).

As previously mentioned, the unfolding rules for simplification and propagation can be obtained as particular cases of Definition 8, by setting $H'_1 = \epsilon$ and $H'_2 = \epsilon$, respectively, and by considering accordingly the resulting unfolded rule.

Example 4

Consider the program P consisting of the following four rules

$$\begin{array}{l} r_1@f(X, Y), f(Y, Z), f(Z, W) \Leftrightarrow g(X, Z)\#1, f(Z, W)\#2, gs(Z, X)\#3 \\ r_2@g(U, V), f(V, T) \Leftrightarrow gg(U, T)\#1 \\ r_3@g(U, V), f(V, T) \Rightarrow gg(U, T)\#1 \\ r_4@g(J, L) \setminus f(L, N) \Leftrightarrow gg(J, N)\#1 \end{array}$$

that we call $cl_{r_1}, cl_{r_2}, cl_{r_3}$, and cl_{r_4} , respectively. This program deduces information about genealogy. Predicate f is considered as father, g as grandfather, gs as grandson and gg as great-grandfather. The following rules are such that we can unfold some constraints in the body of cl_{r_1} using the rule cl_{r_2}, cl_{r_3} , and cl_{r_4} .

³ Here and in the following examples, we use an identifier and also a name for a rule. The reason for this is that after having performed an unfolding we could have different rules labeled by the same identifier. Moreover, we omit the token stores if they are empty.

Now, we unfold the body of rule cl_{r_1} by using the simplification rule cl_{r_2} . We use the *inst* function $inst(gg(U, T)\#1, \emptyset, 3) = (gg(U, T)\#4, \emptyset, 4)$. So the new unfolded rule is:

$$r_1 @ f(X, Y), f(Y, Z), f(Z, W) \Leftrightarrow gs(Z, X)\#3, gg(U, T)\#4, X = U, Z = V, W = T.$$

Now, we unfold the body of cl_{r_1} by using the propagation rule cl_{r_3} . As in the previous case, we have that $inst(gg(U, T)\#1, \emptyset, 3) = (gg(U, T)\#4, \emptyset, 4)$ and then the new unfolded rule is:

$$r_1 @ f(X, Y), f(Y, Z), f(Z, W) \Leftrightarrow g(X, Z)\#1, f(Z, W)\#2, gs(Z, X)\#3, \\ gg(U, T)\#4, X = U, Z = V, W = T; \{r_3 @ 1, 2\}.$$

Finally, we unfold the body of rule cl_{r_1} by using the simpagation rule cl_{r_4} . As before, the function

$$inst(gg(J, N)\#1, \emptyset, 3) = (gg(J, N)\#4, \emptyset, 4)$$

is computed. The new unfolded rule is:

$$r_1 @ f(X, Y), f(Y, Z), f(Z, W) \Leftrightarrow g(X, Z)\#1, gs(Z, X)\#3, \\ gg(J, N)\#4, X = J, Z = L, W = J.$$

The following example considers more specialized rules with guards which are not **true**.

Example 5

Consider the program consisting of the following rules

$$r_1 @ f(X, Y), f(Y, Z), f(Z, W) \Leftrightarrow X = Adam, Y = Seth | \\ g(X, Z)\#1, f(Z, W)\#2, gs(Z, X)\#3, Z = Enosh \\ r_2 @ g(U, V), f(V, T) \Rightarrow U = Adam, V = Enosh | gg(U, T)\#1, T = Kenan \\ r_3 @ g(J, L) \setminus f(L, N) \Leftrightarrow J = Adam, L = Enosh | gg(J, N)\#1, N = Kenan$$

that, as usual, we call cl_{r_1} , cl_{r_2} , and cl_{r_3} , respectively, and which specialize the rules introduced in Example 4 to the genealogy of Adam. That is, here we remember that Adam was father of Seth; Seth was father of Enosh; Enosh was father of Kenan. As before, we consider the predicate f as father, g as grandfather, gs as grandson and gg as great-grandfather.

If we unfold cl_{r_1} by using cl_{r_3} we have

$$r_1 @ f(X, Y), f(Y, Z), f(Z, W) \Leftrightarrow X = Adam, Y = Seth | \\ g(X, Z)\#1, gs(Z, X)\#3, Z = Enosh, \\ gg(J, N)\#4, N = Kenan, X = J, Z = L, W = N.$$

Moreover, when cl_{r_2} is considered to unfold cl_{r_1} , we obtain

$$r_1 @ f(X, Y), f(Y, Z), f(Z, W) \Leftrightarrow X = Adam, Y = Seth | \\ g(X, Z)\#1, f(Z, W)\#2, gs(Z, X)\#3, Z = Enosh, \\ gg(U, T)\#4, T = Kenan, X = U, Z = V, W = T; \{r_2 @ 1, 2\}.$$

Note that $U = Adam$, $V = Enosh$, which is the guard of the rule cl_{r_2} , is not added to the guard of the unfolded rule because $U = Adam$ is entailed by the guard

of cl_{r_1} and $V = Enosh$ is entailed by the built-in constraints in the body of cl_{r_1} , by considering also the binding provided by the parameter passing (analogously for cl_{r_3}).

Example 6

The program P' of the Example 2 is obtained from the program $Ann(P)$ of Example 1 by adding to $Ann(P)$ the clauses resulting from the unfolding of the clause r_2 with r_1 , r_2 and r_3 and from the unfolding of the clause r_4 with r_3 . It is worth noticing that the use of the unfolded clauses allows to decrease the number of Apply transition steps in the successful derivation.

The following result states the correctness of our unfolding rule. The proof is in the Appendix.

Proposition 2

Let P be an annotated CHR program with $cl_r, cl_v \in P$. Let cl'_r be the result of the unfolding of cl_r with respect to cl_v and let P' be the program obtained from P by adding rule cl'_r . Then, for every goal G , $\mathcal{QA}'_{P'}(G) = \mathcal{QA}'_P(G)$ holds.

Since the previous result is independent from the choice of the particular substitution θ which satisfies the conditions of Definition 8, we can choose any such a substitution in order to define the unfolding.

Using the semantic equivalence of a CHR program and its annotated version, we obtain also the following corollary which shows the equivalence between a CHR program and its annotated and unfolded version.

Corollary 1

Let P and $Ann(P)$ be respectively a CHR program and its annotated version. Moreover let $cl_r, cl_v \in Ann(P)$ be CHR annotated rules such that cl'_r is the result of the unfolding of cl_r with respect to cl_v and $P' = Ann(P) \cup \{cl'_r\}$. Then, for every goal G , $\mathcal{QA}_P(G) = \mathcal{QA}'_{P'}(G)$.

Proof

The proof follows from Proposition 1 and Proposition 2. \square

5 Safe rule replacement

The previous result shows that we can safely add to a program P a rule resulting from the unfolding, while preserving the semantics of P in terms of qualified answers. However, when a rule $cl_r \in P$ has been unfolded producing the new rule cl'_r , in some cases we would also like to replace cl_r by cl'_r in P , since this could improve the efficiency of the resulting program. Performing such a replacement while preserving the semantics is in general a very difficult task.

In the case of CHR this is mainly due to three problems. The first one is the presence of guards in the rules. Intuitively, when unfolding a rule r by using a rule v (i.e. when replacing in the body of r a “call” of a procedure by its definition v) it could happen that some guard in v is not satisfied “statically” (i.e. when

performing the unfold), even though it could become satisfied at run-time when the rule v is actually used. If we move the guard of v in the unfolded version of r we can then loose some computations, because the guard in v is moved before the atoms in the body of r (and those atoms could instantiate and satisfy the guard). In other words, the overall guard in the unfolded rule has been strengthened, which means that the rule applies in fewer cases. This implies that if we want to preserve the meaning of a program in general we cannot replace the rule r by its unfolded version. Suitable conditions can be defined in order to allow such a replacement, as we do later. The second source of difficulties consists in the pattern matching mechanism which is used by the CHR computation. According to this mechanism, when rewriting a goal G by a rule r only the variables in the head of r can be instantiated (to become equal to the terms in G). Hence, it could happen that statically the body of a rule r is not instantiated enough to perform the pattern matching involved in the unfolding, while it could become instantiated at run-time in the computations. Also in this case replacing r by its unfolded version in general is not correct. Note that this is not a special case of the first issue, indeed if we cannot (statically) perform the pattern matching we do not unfold the rule while if we move the pattern matching to the guard we could still unfold the rule (under suitable conditions).

Finally, we have the problem of the multiple heads. In fact, let B be the body of a rule r and let H be the (multiple) head of a rule v , which can be used to unfold r : we cannot be sure that at run-time all the atoms in H will be used to rewrite B , since in general B could be in a conjunction with other atoms even though the guards are satisfied. Note that the last point does not mean that the answers of the transformed program are a subset of those of the original one, since by deleting some computations we could introduce in the transformed program new qualified answers which were not in the original program. This is a peculiarity of CHR and it is different from what happens in Prolog.

The next subsection clarifies these three points by using some examples.

5.1 Replacement problems

As previously mentioned, the first problem in replacing a rule by its unfolded version concerns the anticipation of the guard of the rule cl_v (used to unfold the rule cl_r) in the guard of cl_r (as we do in the unfold operation). In fact, as shown by the following example, this could lead to the loss of some computations, when the unfolded rule cl'_r is used rather than the original rule cl_r .

Example 7

Let us consider the program

$$P = \left\{ \begin{array}{l} r_1 @ p(Y) \Leftrightarrow q(Y), s(Y) \\ r_2 @ q(Z) \Leftrightarrow Z = a \mid \mathbf{true} \\ r_3 @ s(V) \Leftrightarrow V = a \end{array} \right\}$$

where we do not consider the identifiers (and the local token store) in the body of rules, because we do not have propagation rules in P .

The unfolding of $r_1@p(Y) \Leftrightarrow q(Y), s(Y)$ by using the rule $r_2@q(Z) \Leftrightarrow Z = a \mid \mathbf{true}$ returns the new rule $r_1@p(Y) \Leftrightarrow Y = a \mid s(Y), Y = Z$. Now the program

$$P' = \left\{ \begin{array}{l} r_1@p(Y) \Leftrightarrow Y = a \mid s(Y), Y = Z \\ r_2@q(Z) \Leftrightarrow Z = a \mid \mathbf{true} \\ r_3@s(V) \Leftrightarrow V = a \end{array} \right\}$$

is not semantically equivalent to P in terms of qualified answers. In fact, given the goal $G = p(X)$ we have $(X = a) \in \mathcal{QA}'_P(G)$, while $(X = a) \notin \mathcal{QA}'_{P'}(G)$.

The second problem is related to the pattern matching used in CHR computations. In fact, following Definition 8, there are some matchings that could become possible only at run-time, and not at compile time, because a stronger (as a first order formula) built-in constraint store is needed. Also in this case, a rule elimination could lead to lose possible answers as illustrated in the following example.

Example 8

Let us consider the program

$$P = \left\{ \begin{array}{l} r_1@p(X, Y) \Leftrightarrow q(Y, X) \\ r_2@q(W, a) \Leftrightarrow W = b \\ r_3@q(J, T) \Leftrightarrow J = d \end{array} \right\}$$

where, as before, we do not consider the identifiers and the token store in the body of rules, because we do not have propagation rules in P . Let P' be the program where the unfolded rule $r_1@p(X, Y) \Leftrightarrow Y = J, X = T, J = d$, obtained by using $r_3@q(J, T) \Leftrightarrow J = d$ in P , substitutes the original one (note that other unfoldings are not possible, in particular the rule $r_2@q(W, a) \Leftrightarrow W = b$ can not be used to unfold $r_1@p(X, Y) \Leftrightarrow q(Y, X)$)

$$P' = \left\{ \begin{array}{l} r_1@p(X, Y) \Leftrightarrow Y = J, X = T, J = d \\ r_2@q(W, a) \Leftrightarrow W = b \\ r_3@q(J, T) \Leftrightarrow J = d \end{array} \right\}.$$

Let $G = p(a, R)$ be a goal. We can see that $(R = b) \in \mathcal{QA}'_P(G)$ and $(R = b) \notin \mathcal{QA}'_{P'}(G)$ because, with the considered goal (and consequently the considered built-in constraint store) $r_2@q(W, a) \Leftrightarrow W = b$ can fire in P but can not fire in P' .

The third problem is related to multiple heads. In fact, the unfolding that we have defined assumes that the head of a rule matches completely with the body of another one, while in general, during a CHR computation, a rule can match with constraints produced by more than one rule and/or introduced by the initial goal. The following example illustrates this point.

Example 9

Let us consider the program

$$P = \left\{ \begin{array}{l} r@p(Y) \Leftrightarrow q(Y), h(b) \\ v@q(Z), h(V) \Leftrightarrow Z = V \end{array} \right\}$$

where we do not consider the identifiers and the token store in the body of rules, as usual.

The unfolding of $r@p(Y) \Leftrightarrow q(Y), h(b)$ by using $v@q(Z), h(V) \Leftrightarrow Z = V$ returns the new rule

$$r@p(Y) \Leftrightarrow Y = Z, V = b, Z = V.$$

Now the program

$$P' = \left\{ \begin{array}{l} r@p(Y) \Leftrightarrow Y = Z, V = b, Z = V \\ v@q(Z), h(V) \Leftrightarrow Z = V \end{array} \right\}$$

where we substitute the original rule by its unfolded version is not semantically equivalent to P . In fact, given the goal $G = p(X), h(a), q(b)$, we have that $(X = a) \in \mathcal{QA}'_P(G)$, while $(X = a) \notin \mathcal{QA}'_{P'}(G)$.

5.2 A condition for safe rule replacement

We have identified some conditions which ensure that we can safely replace the original rule cl_r by its unfolded version while maintaining the qualified answers semantics. Intuitively, this holds when: 1) the constraints of the body of cl_r can be rewritten only by CHR rules such that all the atoms in the head contain the same set of variables; 2) there exists no rule cl_v which can be fired by using a part of constraints introduced in the body of cl_r plus some other constraints; 3) all the rules that can be applied at run-time to the body of the original rule cl_r , can also be applied at transformation time. Before defining formally these conditions, we need some further notations. First of all, given a rule cl_r we define two sets.

The first one contains a set of pairs: for each pair the first component is a rule that can be used to unfold cl_r , while the second one is the sequence of the identifiers of the atoms in the body of cl_r which are used in the unfolding.

The second set contains all the rules that can be used for the *partial unfolding* of cl_r ; in other words, it is the set of rules that can fire by using at least an atom in the body of cl_r and necessarily some other CHR and built-in constraints. Moreover, such a set contains also the rules that can fire, when an opportune built-in constraint store is provided by the computation, but that cannot be unfolded.

Definition 9

Let P be an annotated CHR program and let cl_r, cl_v be the following two annotated rules

$$\begin{array}{l} r@H_1 \setminus H_2 \Leftrightarrow D \mid A; T \text{ and} \\ v@H'_1 \setminus H'_2 \Leftrightarrow D' \mid B; T' \end{array}$$

such that $cl_r, cl_v \in P$ and cl_v is renamed apart with respect to cl_r . We define U^+ and $U^\#$ as follows:

1. $(cl_v, (i_1, \dots, i_n)) \in U^+_P(cl_r)$ if and only if cl_r can be unfolded with cl_v (by Definition 8) by using the sequence of the identified atoms in A with identifiers (i_1, \dots, i_n) .
2. $cl_v \in U^\#_P(cl_r)$ if and only if at least one of the following conditions holds:

- (a) there exists $(A_1, A_2) \subseteq A$ and a built-in constraint C such that

$Fv(C) \cap Fv(cl_v) = \emptyset$, the constraint $D \wedge C$ is satisfiable, $\mathcal{CT} \models (D \wedge C) \rightarrow \exists_{cl_v}((chr(A_1, A_2) = (H'_1, H'_2)) \wedge D')$, $v@id(A_1, A_2) \notin T$, and $(cl_v, id(A_1, A_2)) \notin U_P^+(cl_r)$

- (b) or there exist $k \in A$, $h \in H'_1 \uplus H'_2$ and a built-in constraint C such that $Fv(C) \cap Fv(cl_v) = \emptyset$, the constraint $D \wedge C$ is satisfiable, $\mathcal{CT} \models (D \wedge C) \rightarrow \exists_{cl_v}((chr(k) = h) \wedge D')$, and there exists no $(A_1, A_2) \subseteq A$ such that $v@id(A_1, A_2) \notin T$ and $\mathcal{CT} \models (D \wedge C \wedge (chr(k) = h)) \rightarrow (chr(A_1, A_2) = (H'_1, H'_2))$.

Some explanations are in order here.

The set U^+ contains all the couples composed by those rules that can be used to unfold a fixed rule cl_r , and the identifiers of the constraints considered in the unfolding, introduced in Definition 8.

Let us consider now the set $U^\#$. The conjunction of built-in constraints C represents a generic set of built-in constraints (such a set naturally can be equal to every possible built-in constraint store that can be generated by a real computation before the application of rule cl_v); the condition $Fv(C) \cap Fv(cl_v) = \emptyset$ is required to avoid free variable capture, it represents the renaming (with fresh variables) of a rule cl_v with respect to the computation before the use of the cl_v itself in an **Apply**' transition; the condition $v@id(A_1, A_2) \notin T$ avoids trivial non-termination due the propagation rules; the conditions $\mathcal{CT} \models (D \wedge C) \rightarrow \exists_{cl_v}((chr(A_1, A_2) = (H'_1, H'_2)) \wedge D')$ and $\mathcal{CT} \models (D \wedge C) \rightarrow \exists_{cl_v}((chr(k) = h) \wedge D')$ secure that a strong enough built-in constraint is provided by the computation, before the application of rule cl_v ; finally, the condition $(cl_v, id(A_1, A_2)) \notin U_P^+(cl_r)$ is required to avoid to consider the rules that can be correctly unfolded in the body of cl_r . There are two kinds of rules that are added to $U^\#$. The first one, due to Condition 2a in Definition 9, indicates a matching substitution problem similar to that one described in Example 8. The second kind, due to Condition 2b in Definition 9, indicates a multiple heads problem similar to that one in Example 9. Hence, as we will see in Definition 11, in order to be able to correctly perform the unfolding, the set $U^\#$ must be empty.

Note also that if $U_P^+(cl_r)$ contains a pair, whose first component is a rule with a multiple head and such that the atoms in the head contain different sets of variables, then by definition, $U_P^\#(cl_r) \neq \emptyset$ (Condition 2b of Definition 9).

The following definition introduces a notation for the set obtained by unfolding a rule with (the rules in) a program.

Definition 10

Let P be an annotated CHR program and assume that $cl \in P$,

$$Unf_P(cl)$$

is the set of all annotated rules obtained by unfolding the rule cl with a rule in P , by using Definition 8.

We can now give the central definition of this section.

Definition 11 (SAFE RULE REPLACEMENT)

Let P be an annotated CHR program and let $cl_r \in P$ be the annotated rule $r@H_1 \setminus H_2 \Leftrightarrow D \mid A; T$, such that the following holds

- i) $U_P^\#(cl_r) = \emptyset$,
- ii) $U_P^+(cl_r) \neq \emptyset$ and
- iii) for each $r@H_1 \setminus H_2 \Leftrightarrow D' \mid A'; T' \in Unf_P(cl_r)$ we have that $\mathcal{CT} \models D \leftrightarrow D'$.

Then, we say that the rule cl_r can be safely replaced (by its unfolded version) in P .

Condition **i**) of the previous definition implies that cl_r can be safely replaced in P only if:

- $U_P^+(cl_r)$ contains only pairs, whose first component is a rule such that each atom in the head contains the same set of variables;
- a sequence of identified atoms of body of the rule cl_r can be used to fire a rule cl_v only if cl_r can be unfolded with cl_v by using the same sequence of the identified atoms.

Condition **ii**) states that there exists at least one rule for unfolding the rule cl_r .

Condition **iii**) states that each annotated rule obtained by the unfolding of cl_r in P must have a guard equivalent to that one of cl_r : in fact the condition $\mathcal{CT} \models D \leftrightarrow D'$ in **iii**) avoids the problems discussed in Example 7, thus allows the moving (i.e. strengthening) of the guard in the unfolded rule.

Note that Definition 11 is independent from the particular substitution θ chosen in Definition 8 in order to define the unfolding of the rule

$$\begin{aligned} r@H_1 \setminus H_2 &\Leftrightarrow D \mid K, S_1, S_2, C; T \text{ with respect to} \\ v@H'_1 \setminus H'_2 &\Leftrightarrow D' \mid B; T' \end{aligned}$$

In fact, let us assume that there exist two substitution θ and γ which satisfy the conditions of Definition 8. Then $\mathcal{CT} \models (C \wedge D) \rightarrow (d\theta \leftrightarrow d\gamma)$ for each $d \in D'$. Therefore, if $V = \{d \in D' \mid \mathcal{CT} \models C \wedge D \rightarrow d\theta\}$ and $W = \{d \in D' \mid \mathcal{CT} \models C \wedge D \rightarrow d\gamma\}$, we have that $V = W$ and then $D'' = D \setminus V = D \setminus W$. Now, it is easy to check that Condition **iii**) follows if and only if $D''\theta = D''\gamma = \emptyset$.

The following is an example of a safe replacement.

Example 10

Consider the program P consisting of the following four rules

$$\begin{aligned} r_1@p(X, Y, Z) &\Leftrightarrow r(b, b, Z)\#1, s(Z, b, a)\#2, q(X, f(Z), a)\#3, r(g(X, b), f(a), f(Z))\#4; \emptyset \\ r_2@q(V, U, W), r(g(V, b), f(W), U) &\Leftrightarrow W = a \mid s(V, U, W)\#1, r(U, U, V)\#2; \emptyset \\ r_3@r(M, M, N), s(N, M, a) &\Leftrightarrow p(M, N, N)\#1; \emptyset \\ r_4@s(L, J, I) &\Rightarrow I = L; \emptyset \end{aligned}$$

where the four rules identified by r_1, r_2, r_3 , and r_4 are called $cl_{r_1}, cl_{r_2}, cl_{r_3}$, and cl_{r_4} , respectively. By Definition 9, we have that

$$\begin{aligned} U_P^+(cl_{r_1}) &= \{cl_{r_2}@3, 4, cl_{r_3}@1, 2, cl_{r_4}@2\} \\ U_P^\#(cl_{r_1}) &= \emptyset. \end{aligned}$$

Moreover

$$\begin{aligned}
Unf_P(cl_{r_1}) = & \\
\{ & r_1 @ p(X, Y, Z) \Leftrightarrow r(b, b, Z) \# 1, s(Z, b, a) \# 2, s(V, U, W) \# 5, r(U, U, V) \# 6, \\
& \quad X = V, U = f(Z), W = a; \emptyset \\
& r_1 @ p(X, Y, Z) \Leftrightarrow q(X, f(Z), a) \# 3, r(g(X, b), f(a), f(Z)) \# 4, p(M, N, N) \# 5, \\
& \quad M = b, N = Z; \emptyset \\
& r_1 @ p(X, Y, Z) \Leftrightarrow r(b, b, Z) \# 1, s(Z, b, a) \# 2, q(X, f(Z), a) \# 3, \\
& \quad r(g(X, b), f(a), f(Z)) \# 4, I = L, Z = L, b = J, a = I; \{cl_{r_4} @ 2\} \quad \}
\end{aligned}$$

Then cl_1 can be safely replaced in P according to Definition 11 and then we obtain

$$P_1 = (P \setminus \{cl_1\}) \cup Unf_P(cl_1),$$

where P_1 is the program

$$\begin{aligned}
\{ & r_1 @ p(X, Y, Z) \Leftrightarrow r(b, b, Z) \# 1, s(Z, b, a) \# 2, s(V, U, W) \# 5, r(U, U, V) \# 6, \\
& \quad X = V, U = f(Z), W = a; \emptyset \\
& r_1 @ p(X, Y, Z) \Leftrightarrow q(X, f(Z), a) \# 3, r(g(X, b), f(a), f(Z)) \# 4, p(M, N, N) \# 5, \\
& \quad M = b, N = Z; \emptyset \\
& r_1 @ p(X, Y, Z) \Leftrightarrow r(b, b, Z) \# 1, s(Z, b, a) \# 2, q(X, f(Z), a) \# 3, \\
& \quad r(g(X, b), f(a), f(Z)) \# 4, I = L, Z = L, b = J, a = I; \{cl_{r_4} @ 2\} \\
& r_2 @ q(V, U, W), r(g(V, b), f(W), U) \Leftrightarrow W = a \mid s(V, U, W) \# 1, r(U, U, V) \# 2; \emptyset \\
& r_3 @ r(M, M, N), s(N, M, a) \Leftrightarrow p(M, N, N) \# 1; \emptyset \\
& r_4 @ s(L, J, I) \Rightarrow I = L; \emptyset \quad \}
\end{aligned}$$

We can now provide the result which shows the correctness of our safe replacement rule. The proof is in the Appendix.

Theorem 1

Let P be an annotated program, cl be a rule in P such that cl can be safely replaced in P according to Definition 11. Assume also that

$$P' = (P \setminus \{cl\}) \cup Unf_P(cl).$$

Then $\mathcal{QA}'_P(G) = \mathcal{QA}'_{P'}(G)$ for any arbitrary goal G .

Of course, the previous result can be applied to a sequence of program transformations. Let us define such a sequence as follows.

Definition 12 (U-SEQUENCE)

Let P be an annotated CHR program. An U -sequence of programs starting from P is a sequence of annotated CHR programs P_0, \dots, P_n , such that

$$\begin{aligned}
P_0 &= P \text{ and} \\
P_{i+1} &= (P_i \setminus \{cl^i\}) \cup Unf_{P_i}(cl^i),
\end{aligned}$$

where $i \in [0, n-1]$, $cl^i \in P_i$ and can be safely replaced in P_i .

Example 11

Let us to consider the program P_1 of Example 10. The clause cl_2 can be safely replaced in P_1 according to Definition 11 and then we obtain

$$P_2 = (P_1 \setminus \{cl_2\}) \cup Unf_{P_1}(cl_2),$$

where P_2 is the program

$$\left. \begin{aligned} & \{ r_1 @ p(X, Y, Z) \Leftrightarrow r(b, b, Z) \# 1, s(Z, b, a) \# 2, s(V, U, W) \# 5, r(U, U, V) \# 6, \\ & \quad X = V, U = f(Z), W = a; \emptyset \\ & r_1 @ p(X, Y, Z) \Leftrightarrow q(X, f(Z), a) \# 3, r(g(X, b), f(a), f(Z)) \# 4, p(M, N, N) \# 5, \\ & \quad M = b, N = Z; \emptyset \\ & r_1 @ p(X, Y, Z) \Leftrightarrow r(b, b, Z) \# 1, s(Z, b, a) \# 2, q(X, f(Z), a) \# 3, \\ & \quad r(g(X, b), f(a), f(Z)) \# 4, I = L, Z = L, b = J, a = I; \{cl_{r_4} @ 2\} \\ & r_2 @ q(V, U, W), r(g(V, b), f(W), U) \Leftrightarrow W = a \mid p(M, N, N) \# 3, V = N, U = M; \emptyset \\ & r_2 @ q(V, U, W), r(g(V, b), f(W), U) \Leftrightarrow W = a \mid s(V, U, W) \# 1, r(U, U, V) \# 2, V = L, \\ & \quad U = J, W = I, I = L; \{cl_{r_4} @ 1\} \\ & r_3 @ r(M, M, N), s(N, M, a) \Leftrightarrow p(M, N, N) \# 1; \emptyset \\ & r_4 @ s(L, J, I) \Rightarrow I = L; \emptyset \end{aligned} \right\}.$$

Then, from Theorem 1 and Proposition 1, we have the following.

Corollary 2

Let P be a program and let P_0, \dots, P_n be an U-sequence starting from $Ann(P)$. Then $\mathcal{QA}_P(G) = \mathcal{QA}'_{P_n}(G)$ for any arbitrary goal G .

5.3 Confluence and Termination

In this section, we prove that our unfolding preserves termination provided that one considers normal derivations. These are the derivations in which the **Solve** (**Solve'**) transitions are applied as soon as possible, as specified by Definition 14. Moreover, we prove that our unfolding preserves also confluence, provided that one considers only non-recursive unfoldings.

We first need to introduce the concept of built-in free configuration: This is a configuration which has no built-in constraints in the first component.

Definition 13 (BUILT-IN FREE CONFIGURATION)

Let $\sigma = \langle G, S, D, T \rangle_o \in Conf_t$ ($\sigma = \langle G, D, T \rangle_o \in Conf'_t$). The configuration σ is built-in free if G is a multiset of (identified) CHR-constraints.

Now, we can introduce the concept of normal derivation.

Definition 14 (NORMAL DERIVATION)

Let P be a (possibly annotated) CHR program and let δ be a derivation in P . We say that δ is normal if, for each configuration σ in δ , a transition **Apply** (**Apply'**) is used on σ only if σ is built-in free.

Note that, by definition, given a CHR program P , $\mathcal{QA}(P)$ can be calculated by considering only normal derivations and analogously for an annotated CHR program P' .

Definition 15 (NORMAL TERMINATION)

A CHR program P is called *terminating*, if there are no infinite derivations. A (possibly annotated) CHR program P is called *normally terminating*, if there are no infinite normal derivations.

The following result shows that normal termination is preserved by unfolding with the safe replacement condition. The proof is in the Appendix.

Proposition 3 (NORMAL TERMINATION)

Let P be a CHR program and let P_0, \dots, P_n be an U-sequence starting from $\text{Ann}(P)$. P satisfies normal termination if and only if P_n satisfies normal termination.

When standard termination is considered rather than normal termination, the previous result does not hold, due to the guard elimination in the unfolding. This is shown by the following example.

Example 12

Let us consider the following program:

$$P = \{ \begin{array}{l} r_1@p(X) \Leftrightarrow X = a, q(X) \\ r_2@q(Y) \Leftrightarrow Y = a \mid r(Y) \\ r_3@r(Z) \Leftrightarrow Z = d \mid p(Z) \end{array} \}$$

where we do not consider the identifiers and the token store in the body of rules (because we do not have propagation rules in P). Then, by using

$$r_2@q(Y) \Leftrightarrow Y = a \mid r(Y)$$

to unfold $r_1@p(X) \Leftrightarrow X = a, q(X)$ (with replacement) we obtain the following program P' :

$$P' = \{ \begin{array}{l} r_1@p(X) \Leftrightarrow X = a, X = Y, r(Y) \\ r_2@q(Y) \Leftrightarrow Y = a \mid r(Y) \\ r_3@r(Z) \Leftrightarrow Z = d \mid p(Z) \end{array} \}.$$

It is easy to check that the program P satisfies the (standard) termination. On the other hand, considering the program P' and the start goal $(V = d, p(V))$, the following state can be reached

$$\langle (X = a, p(Z)\#3), (V = d, V = X, X = Y, Y = Z), \emptyset \rangle_3$$

where rules $r_1@p(X) \Leftrightarrow X = a, X = Y, r(Y)$ and $r_3@r(Z) \Leftrightarrow Z = d \mid p(Z)$ can be applied infinitely many times if the built-in constraint $X = a$ is not moved by the **Solve**' rule into the built-in store. Hence, we have non-termination.

The next property we consider is confluence. This property guarantees that any computation for a goal results in the same final state, no matter which of the applicable rules are applied (Abdennadher and Frühwirth 2004; Frühwirth 2005).

We first give the following definition which introduces some specific notation for renamings of indexes.

Definition 16

Let j_1, \dots, j_o be distinct identification values.

- A renaming of identifiers is a substitution of the form $[j_1/i_1, \dots, j_o/i_o]$, where i_1, \dots, i_o is a permutation of j_1, \dots, j_o .
- Given an expression E and a renaming of identifiers $\rho = [j_1/i_1, \dots, j_o/i_o]$, $E\rho$ is defined as the expression obtained from E by substituting each occurrence of the identification value j_l with the corresponding i_l , for $l \in [1, o]$

- If ρ and ρ' are renamings of identifiers, then $\rho\rho'$ denotes the renaming of identifiers such that for each expression E , $E(\rho\rho') = (E\rho)\rho'$.

We will use ρ, ρ', \dots to denote renamings.

Now, we need the following definition introducing a form of equivalence between configurations, which is a slight modification of that one in (Raiser et al. 2009), since considers a different form of configuration and, in particular, also the presence of the token store. Two configurations are equivalent if they have the same logical reading and the same rules are applicable to these configurations with the same results. By an abuse of notation, when it is clear from the context, we will write \equiv_V to denote two equivalence relations in $Conf_t$ and in $Conf'_t$ with the same meaning.

Definition 17

Let V be a set of variables The equivalence \equiv_V between configurations in $Conf_t$ is the smallest equivalence relation that satisfies the following conditions.

- $\langle d \wedge G, S, C, T \rangle_n \equiv_V \langle G, S, d \wedge C, T \rangle_n$,
- $\langle G, S, X = t \wedge C, T \rangle_n \equiv_V \langle G[X/t], S[X/t], X = t \wedge C, T \rangle_n$,
- Let X, Y be variables such that $X, Y \notin V$ and Y does not occur in G, S or C .
 $\langle G, S, C, T \rangle_n \equiv_V \langle G[X/Y], S[X/Y], C[X/Y], T \rangle_n$,
- If $W = Fv(C) \setminus (Fv(G, S) \cup V)$, $U = Fv(C') \setminus (Fv(G, S) \cup V)$, and $CT \models \exists_W C \leftrightarrow \exists_U C'$ then $\langle G, S, C, T \rangle_n \equiv_V \langle G, S, C', T \rangle_n$,
- $\langle G, S, \mathbf{false}, T \rangle_n \equiv_V \langle G', S', \mathbf{false}, T' \rangle_m$,
- $\langle G, S, C, T \rangle_n \equiv_V \langle G, S\rho, C, T\rho \rangle_m$ for each renaming of identifiers ρ such that for each $i \in id(S\rho) \cup id(T\rho)$ we have that $i < m$,
- $\langle G, S, C, T \rangle_n \equiv_V \langle G, S, C, \mathit{clean}(S, T) \rangle_n$.

We can define the equivalence \equiv_V between configurations in $Conf'_t$ in an analogous way.

Definition 18

Let V be a set of variables The equivalence \equiv_V between configurations in $Conf'_t$ is the smallest equivalence relation that satisfies the following conditions.

- $\langle d \wedge G, C, T \rangle_n \equiv_V \langle G, d \wedge C, T \rangle_n$,
- $\langle G, X = t \wedge C, T \rangle_n \equiv_V \langle G[X/t], X = t \wedge C, T \rangle_n$,
- Let X, Y be variables such that $X, Y \notin V$ and Y does not occur in G or C .
 $\langle G, C, T \rangle_n \equiv_V \langle G[X/Y], C[X/Y], T \rangle_n$,
- If $W = Fv(C) \setminus (Fv(G) \cup V)$, $U' = Fv(C') \setminus (Fv(G) \cup V)$, and $CT \models \exists_W C \leftrightarrow \exists_{U'} C'$ then $\langle G, C, T \rangle_n \equiv_V \langle G, C', T \rangle_n$,
- $\langle G, \mathbf{false}, T \rangle_n \equiv_V \langle G', \mathbf{false}, T' \rangle_m$,
- $\langle G, C, T \rangle_n \equiv_V \langle G\rho, C, T\rho \rangle_m$ for each renaming of identifiers ρ such that for each $i \in id(G\rho) \cup id(T\rho)$ we have that $i \leq m$,
- $\langle G, C, T \rangle_n \equiv_V \langle G, C, \mathit{clean}(G, T) \rangle_n$.

By definition of \equiv_V , it is straightforward to check that if $\sigma, \sigma' \in Conf'_t(Conf'_t)$, V is a set of variables, and $\sigma \equiv_V \sigma'$ then the following holds

- if $W \subseteq V$ then $\sigma \equiv_W \sigma'$ and

- if $X \notin Fv(\sigma) \cup Fv(\sigma')$ then $\sigma \equiv_{V \cup \{X\}} \sigma'$.

We now introduce the concept of confluence which is a slight modification of that one in (Raiser et al. 2009), since it considers also the cleaning of the token store.

In the following \mapsto^* means either $\longrightarrow_{\omega_t}^*$ or $\longrightarrow_{\omega'_t}^*$.

Definition 19 (CONFLUENCE)

A CHR [annotated] program is *confluent* if for any state σ the following holds: if $\sigma \mapsto^* \sigma_1$ and $\sigma \mapsto^* \sigma_2$ then there exist states σ'_f and σ''_f such that $\sigma_1 \mapsto^* \sigma'_f$ and $\sigma_2 \mapsto^* \sigma''_f$, where $\sigma'_f \equiv_{Fv(\sigma)} \sigma''_f$.

Now, we prove that our unfolding preserves confluence, provided that one considers only non-recursive unfolding. These are the unfoldings such that a clause cl cannot be used in order to unfold cl itself.

When safe rule replacement is considered rather than non-recursive safe rule replacement (see Definition 20), the confluence is not preserved. This is shown by the following example.

Example 13

Let us consider the following program:

$$P = \left\{ \begin{array}{l} r_1 @ p \Leftrightarrow q \\ r_2 @ p \Leftrightarrow r \\ r_3 @ r \Leftrightarrow r, s \\ r_4 @ q \Leftrightarrow r, s \end{array} \right\}$$

where we do not consider the identifiers and the token store in the body of rules (because we do not have propagation rules in P). Then, by using r_3 to unfold r_3 itself (with safe rule replacement) we obtain the following program P' :

$$P' = \left\{ \begin{array}{l} r_1 @ p \Leftrightarrow q \\ r_2 @ p \Leftrightarrow r \\ r_3 @ r \Leftrightarrow r, s, s \\ r_4 @ q \Leftrightarrow r, s \end{array} \right\}.$$

It is easy to check that the program P is confluent. On the other hand, considering the program P' and the start goal p , the following two states can be reached

$$\sigma = \langle (r\#3, s\#4, s\#5), \mathbf{true}, \emptyset \rangle_5 \text{ and } \sigma' = \langle (r\#3, s\#4), \mathbf{true}, \emptyset \rangle_4$$

and there exist no states σ_1 and σ'_1 such that $\sigma \longrightarrow_{\omega_t}^* \sigma_1$ and $\sigma' \longrightarrow_{\omega'_t}^* \sigma'_1$ in P' , where $\sigma_1 \equiv_{\emptyset} \sigma'_1$.

Note that the program in previous example is not terminating. We cannot consider a terminating program here, since for such a program (weak) safe rule replacement would allow to preserve confluence. Now, we give the definition of non-recursive safe rule replacement.

Definition 20 (NON-RECURSIVE SAFE RULE REPLACEMENT)

Let P be an annotated CHR program and let $cl_r \in P$ be an annotated rule such that cl_r can be safely replaced (by its unfolded version) in P . We say that cl_r

can be non-recursively safely replaced (by its unfolded version) in P if for each $(cl_v, (i_1, \dots, i_n)) \in U_P^+(cl_r)$, we have that $cl_v \neq cl_r$.

The following is the analogous of Definition 12, where non-recursive safe rule replacement is considered.

Definition 21 (NRU-SEQUENCE)

Let P be an annotated CHR program. An *NRU-sequence* of programs starting from P is a sequence of annotated CHR programs P_0, \dots, P_n , such that

$$\begin{aligned} P_0 &= P \text{ and} \\ P_{i+1} &= (P_i \setminus \{cl^i\}) \cup Unf_{P_i}(cl^i), \end{aligned}$$

where $i \in [0, n-1]$, $cl^i \in P_i$ and can be non-recursively safely replaced in P_i .

Theorem 2

Let P be a CHR program and let P_0, \dots, P_n be an NRU-sequence starting from $P_0 = Ann(P)$. P satisfies confluence if and only if P_n satisfies confluence too.

6 Weak safe rule replacement

In this subsection, we consider only programs which are normally terminating and confluent. For this class of programs we give a condition for rule replacement which is much weaker than that one used in the previous section and which still allows one to preserve the qualified answers semantics. Intuitively this new condition requires that there exists a rule obtained by the unfolding of cl_r in P whose guard is equivalent to that one in cl_r .

Definition 22 (WEAK SAFE RULE REPLACEMENT)

Let P be an annotated CHR program and let $r@H_1 \setminus H_2 \Leftrightarrow D \mid A; T \in P$ be a rule such that there exists

$$r@H_1 \setminus H_2 \Leftrightarrow D' \mid A'; T' \in Unf_P(r@H_1 \setminus H_2 \Leftrightarrow D \mid A; T)$$

with $\mathcal{CT} \models D \Leftrightarrow D'$.

Then, we say that the rule $r@H_1 \setminus H_2 \Leftrightarrow D \mid A; T$ can be weakly safely replaced (by its unfolded version) in P .

Example 14

Let us consider the following program P :

$$P_1 = \left\{ \begin{array}{l} r_1@p(X) \Leftrightarrow q(X), s(X) \\ r_2@t(a) \Leftrightarrow r(b) \\ r_3@q(Y) \Leftrightarrow t(Y) \\ r_4@s(a) \setminus q(a) \Leftrightarrow r(b) \end{array} \right\}$$

where we do not consider the identifiers and the token store in the body of rules (because we do not have propagation rules in P). By Definition 22, r_1 can be weakly safely replaced (by its unfolded version) in P and then we can obtain the program

$$P_1 = (P \setminus \{r_1\}) \cup Unf_P(r_1),$$

where P_1 is the program

$$P_1 = \{ \begin{array}{l} r_1 @ p(X) \Leftrightarrow s(X), t(Y), X = Y \\ r_2 @ t(a) \Leftrightarrow r(b) \\ r_3 @ q(Y) \Leftrightarrow t(Y) \\ r_4 @ s(a) \setminus q(a) \Leftrightarrow r(b) \end{array} \}.$$

Finally, observe that r_1 cannot be safely replaced (by its unfolded version) in P .

The following proposition shows that normal termination and confluence are preserved by weak safe rule replacement. The proof is in the Appendix.

Proposition 4

Let P be an annotated CHR program and let $cl \in P$ such that cl can be weakly safely replaced in P . Moreover let

$$P' = (P \setminus \{cl\}) \cup Unf_P(cl).$$

If P is normally terminating then P' is normally terminating. If P is normally terminating and confluent then P' is confluent too.

The converse of the previous theorem does not hold, as shown by the following example.

Example 15

Let us consider the following program:

$$P = \{ \begin{array}{l} r_1 @ p(X) \Leftrightarrow q(X) \\ r_2 @ q(a) \Leftrightarrow p(a) \\ r_3 @ q(Y) \Leftrightarrow r(Y) \end{array} \}$$

where we do not consider the identifiers and the token store in the body of rules (because we do not have propagation rules in P). Then, by using r_3 to unfold r_1 itself (with weak safe rule replacement) we obtain the following program P' :

$$P' = \{ \begin{array}{l} r_1 @ p(X) \Leftrightarrow X = Y, r(Y) \\ r_2 @ q(a) \Leftrightarrow p(a) \\ r_3 @ q(Y) \Leftrightarrow r(Y). \end{array} \}$$

It is easy to check that the program P' satisfies the (normal) termination. On the other hand, considering the program P and the start goal $p(a)$, the following state can be reached

$$\langle (p(a) \# 3), (X = a), \emptyset \rangle_3$$

where rules $r_1 @ p(X) \Leftrightarrow q(X)$ and $r_2 @ q(a) \Leftrightarrow p(a)$ in P can be applied infinitely many times. Hence, we have non-(normally)termination.

Next, we show that weak safe rule replacement transformation preserves qualified answers.

Theorem 3

Let P be a normally terminating and confluent annotated program and let cl be a rule in P such that cl can be weakly safely replaced in P according to Definition 22. Assume also that

$$P' = (P \setminus \{cl\}) \cup Unf_P(cl).$$

Then $\mathcal{QA}'_P(G) = \mathcal{QA}'_{P'}(G)$ for any arbitrary goal G .

Proof

Analogously to Theorem 1, by using Proposition 1 we can prove that $\mathcal{QA}'_P(G) = \mathcal{QA}'_{P''}(G)$ where

$$P'' = P \cup Unf_P(cl),$$

for any arbitrary goal G .

Then, to prove the thesis, we have only to prove that

$$\mathcal{QA}'_{P'}(G) = \mathcal{QA}'_{P''}(G).$$

We prove the two inclusions separately.

$(\mathcal{QA}'_{P'}(G) \subseteq \mathcal{QA}'_{P''}(G))$ The proof is the same of the case $\mathcal{QA}'_{P'}(G) \subseteq \mathcal{QA}'_{P''}(G)$ of Theorem 1 and hence it is omitted.

$(\mathcal{QA}'_{P''}(G) \subseteq \mathcal{QA}'_{P'}(G))$ The proof is by contradiction. Assume that there exists $Q \in \mathcal{QA}'_{P''}(G) \setminus \mathcal{QA}'_{P'}(G)$. Since, from the proof of Proposition 4, we can conclude that P'' is normally terminating and confluent, we have that $\mathcal{QA}'_{P''}(G)$ is a singleton. Moreover, since by the previous point $\mathcal{QA}'_{P'}(G) \subseteq \mathcal{QA}'_{P''}(G)$, we have that $\mathcal{QA}'_{P'}(G) = \emptyset$. This means that each normal derivation in P' either is not terminating or terminates with a failed configuration. Then, by using Proposition 4, we have that each normal derivation in P' terminates with a failed configuration. Since $P' \subseteq P''$, we have that there exist normal derivations in P'' which terminate with a failed configuration. Then, by Lemma 3 and since $Q \in \mathcal{QA}'_{P''}(G)$, we have a contradiction and then the thesis holds.

□

Let cl be the rule $r@H_1 \setminus H_2 \Leftrightarrow D \mid A; T$. Note that Proposition 4 and Theorem 3 hold also if

$$P' = (P \setminus \{cl\}) \cup S,$$

where $S \subseteq Unf_P(cl)$ and there exists $cl' = r@H_1 \setminus H_2 \Leftrightarrow D' \mid A'; T' \in S$ such that $\mathcal{CT} \models D \leftrightarrow D'$.

If in Definition 12 we consider weak safe rule replacement rather than safe rule replacement, then we can obtain a definition of WU-sequence (rather than U-sequence). From the previous theorem and by Proposition 4, by using an obvious inductive argument, we can derive that the semantics (in terms of qualified answers) is preserved in WU-sequences starting from a normally terminating and confluent annotated program, where weak safe replacement is applied repeatedly.

7 Conclusions

In this paper, we have defined an unfold operation for CHR which preserves the qualified answers of a program.

This was obtained by transforming a CHR program into an annotated one which is then unfolded. The equivalence of the unfolded program and the original (unannotated) one is proven by using a slightly modified operational semantics for annotated programs. We have then provided a condition that can be used to replace a rule by its unfolded version, while preserving the qualified answers. We have also shown that this condition ensures that confluence and termination are preserved, provided that one considers normal derivations. Finally, we have defined a further, weaker, condition which allows one to safely replace a rule by its unfolded version (while preserving qualified answers) for programs which are normally terminating and confluent.

There are only few other papers that consider source to source transformation of CHR programs. (Frühwirth 2005), rather than considering a generic transformation system focuses on the specialization of rules w.r.t. a specific goal, analogously to what happens in partial evaluation. In (Frühwirth and Holzbaur 2003) CHR rules are transformed in a relational normal form, over which a source to source transformation is performed. Some form of transformation for probabilistic CHR is considered in (Frühwirth et al. 2002), while guard optimization was studied in (Sneyers et al. 2005). Another paper which involves program transformation for CHR is (Sarna-Starosta and Schrijvers 2009).

Both the general and the goal specific approaches are important in order to define practical transformation systems for CHR. In fact, on the one hand of course one needs some general unfold rule, on the other hand, given the difficulties in removing rules from the transformed program, some goal specific techniques can help to improve the efficiency of the transformed program for specific classes of goals. A method for deleting redundant CHR rules is considered in (Abdennadher and Frühwirth 2004). However, it is based on a semantic check and it is not clear whether it can be transformed into a specific syntactic program transformation rule.

When considering more generally the field of concurrent logic languages, we find few papers which address the issue of program transformation. Notable examples include (Etalle et al. 2001) that deals with the transformation of Concurrent Constraint Programming (CCP) and (Ueda and Furukawa 1988) that considers Guarded Horn Clauses (GHC). The results in these papers are not directly applicable to CHR because neither CCP nor GHC allow rules with multiple heads.

As mentioned in the introduction, some of the results presented here appeared in (Tacchella et al. 2007) and in the thesis (Tacchella 2008). However, it is worth noticing that the conditions for safe rule replacement that we have presented in Section 5 and the content of Section 6 are original contributions of this paper. In particular, differently from the conditions given in (Tacchella et al. 2007) and (Tacchella 2008), the conditions defined in Section 5 allow us to perform rule re-

placement also when rules with multiple heads are used for unfolding a given rule. This is a major improvement, since CHR rules have naturally multiple heads.

The results obtained in the current article can be considered as a first step in the direction of defining a transformation system for CHR programs, based on unfolding. This step could be extended in several directions: First of all, the unfolding operation could be extended to take into account also the constraints in the propagation part of the head of a rule. Also, we could extend to CHR some of the other transformations, notably folding (Tamaki and Sato 1984) which has already been applied to CCP in (Etalle et al. 2001). Finally, we would like to investigate from a practical perspective to what extent program transformation can improve the performance of the CHR solver. Clearly, the application of an unfolded rule avoids some computational steps (assuming that unfolding is done at the time of compilation, of course). However, the increase in the number of program rules produced by unfolding could eliminate this improvement.

Here, it would probably be important to consider some unfolding strategy, in order to decide which rules have to be unfolded.

An efficient unfolding strategy could also incorporate in particular probabilistic or statistical information. The idea would be to only unfold CHR rules which are used often and leave those which are used only occasionally unchanged in order to avoid an unnecessary increase in the number of program rules. This approach could be facilitated by probabilistic CHR extensions such as the ones as presented for example in (Frühwirth et al. 2002) and (Sneyers et al. 2010). Extending the results of this paper to probabilistic CHR will basically follow the lines and ideas presented here. The necessary information which one would need to decide whether and in which sequence to unfold CHR rules could obtain experimentally, e.g. by profiling, or formally via probabilistic program analysis. One could see this as a kind of *speculative* unfolding.

References

- ABDENNADHER, S. 1997. Operational semantics and confluence of constraint propagation rules. In *Principles and Practice of Constraint Programming-CP97*, G. Smolka, Ed. Lecture Notes in Computer Science, vol. 1330. Springer, Berlin / Heidelberg, 252–266.
- ABDENNADHER, S. AND FRÜHWIRTH, T. 2004. Integration and organization of rule-based constraint solvers. In *Logic Based Program Synthesis and Transformation (LOPSTR 2003)*, M. Bruynooghe, Ed. Lecture Notes in Computer Science, vol. 3018/2004. Springer-Verlag, Berlin / Heidelberg, 198–231.
- DUCK, G. J., STUCKEY, P. J., GARCA DE LA BANDA, M., AND HOLZBAUR, C. 2004. The refined operational semantics of constraint handling rules. In *Logic Programming*, B. Demoen and V. Lifschitz, Eds. Lecture Notes in Computer Science, vol. 3132. Springer, Berlin / Heidelberg, 269–304.
- ETALLE, S., GABBRIELLI, M., AND MEO, M. C. 2001. Transformations of CCP programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 23, 3, 304–395.
- FRÜHWIRTH, T. 1998. Theory and Practice of Constraint Handling Rules. *Journal of Logic Programming* 37, 1-3 (October), 95–138. Special Issue on Constraint Logic Programming.
- FRÜHWIRTH, T. 2005. Specialization of concurrent guarded multi-set transformation rules.

- In *Logic Based Program Synthesis and Transformation*, S. Etalle, Ed. Lecture Notes in Computer Science, vol. 3573. Springer, Berlin / Heidelberg, 133–148.
- FRÜHWIRTH, T. 2009. *Constraint Handling Rules*. Cambridge University Press, Cambridge.
- FRÜHWIRTH, T. AND ABDENNADHER, S. 2003. *Essentials of Constraint Programming*. Springer, Berlin / Heidelberg.
- FRÜHWIRTH, T., DI PIERRO, A., AND WIKLICKY, H. 2002. Probabilistic constraint handling rules. In *Selected Papers of WFLP'02 — 11th International Workshop on Functional and (Constraint) Logic Programming*, M. Comini and M. Falaschi, Eds. Electronic Notes in Theoretical Computer Science, vol. 76. Elsevier.
- FRÜHWIRTH, T. AND HOLZBAUR, C. 2003. Source-to-source transformation for a class of expressive rules. In *APPIA-GULP-PRODE*. Pub Zone, Reggio Calabria, Italy, 386–397.
- LLOYD, J. W. 1984. *Foundations of logic programming*. Springer-Verlag New York, Inc., New York, NY, USA.
- RAISER, F., BETZ, H., AND FRÜHWIRTH, T. 2009. Equivalence of CHR states revisited. In *Proceedings of the 6th International Workshop on Constraint Handling Rules (CHR)*. 34–48.
- SARNA-STAROSTA, B. AND SCHRIJVERS, T. 2009. Attributed Data for CHR Indexing. In *ICLP*. 357–371.
- SCHRIJVERS, T. AND SULZMANN, M. 2008. Transactions in constraint handling rules. In *ICLP*. 516–530.
- SNEYERS, J., MEERT, W., VENNEKENS, J., KAMEYA, Y., AND SATO, T. 2010. CHR(PRISM)-based probabilistic logic learning. *Theory and Practice of Logic Programming* 10, 4-6, 433–447.
- SNEYERS, J., SCHRIJVERS, T., AND DEMOEN, B. 2005. Guard and continuation optimization for occurrence representations of CHR. In *21st International Conference, ICLP 2005*, M. Gabbrielli and G. Gupta, Eds. Lecture Notes in Computer Science, vol. 3668. Springer, Berlin / Heidelberg, 83–97.
- TACCHELLA, P. 2008. Constraint handling rules. Compositional semantics and program transformation. Ph.D. thesis, Università di Bologna.
- TACCHELLA, P., MEO, M. C., AND GABBRIELLI, M. 2007. Unfolding in CHR. In *PPDP '07: Proceedings of the 9th ACM SIGPLAN international symposium on Principles and Practice of Declarative Programming*. ACM, New York, NY, USA, 179–186.
- TAMAKI, H. AND SATO, T. 1984. Unfold/Fold transformations of logic programs. In *Proceedings of International Conference on Logic Programming*. Uppsala University, Uppsala, Sweden, 127–138.
- UEDA, K. AND FURUKAWA, K. 1988. Transformation rules for GHC programs. In *Proceedings of International Conference on Fifth Generation Computer Systems 1988 (FGCS'88)*. ICOT Press, Tokyo, Japan, 582–591.

Appendix A Proofs

In this appendix, we give the proofs of some of the results contained in the paper.

A.1 Equivalence of the two operational semantics

Here, we provide the proof of Proposition 1. To this aim we first introduce some preliminary notions and lemmas.

Then, we define two configurations (in the two different transition systems) equivalent when they are essentially the same up to renaming of identifiers.

Definition 23 (CONFIGURATION EQUIVALENCE)

Let $\sigma = \langle (H_1, C), H_2, D, T \rangle_n \in \text{Conf}_t$ be a configuration in the transition system ω_t and let $\sigma' = \langle (K, C), D, T' \rangle_m \in \text{Conf}'_t$ be a configuration in the transition system ω'_t .

σ and σ' are *equivalent* (and we write $\sigma \approx \sigma'$) if:

1. there exist K_1 and K_2 , such that $K = K_1 \uplus K_2$, $H_1 = \text{chr}(K_1)$ and $\text{chr}(H_2) = \text{chr}(K_2)$,
2. for each $l \in \text{id}(K_1)$, l does not occur in T' ,
3. there exists a renaming of identifier ρ s.t. $H_2\rho = K_2$ and $T\rho = T'$.

Condition 1 grants that σ and σ' have equal CHR constraints, while Condition 2 ensures that no propagation rule is applied to constraints in σ' corresponding to constraints in σ that are not previously introduced in the CHR store. Finally, condition 3 requires that there exists a renaming of identifiers such that the identified CHR constraints and the tokens of σ and the ones associated with them in σ' are equal, after the renaming.

The following result shows the equivalence of the two introduced semantics proving the equivalence of intermediate configurations.

Lemma 1

Let P and $\text{Ann}(P)$ be respectively a CHR program and its annotated version. Moreover, let $\sigma \in \text{Conf}_t$ and let $\sigma' \in \text{Conf}'_t$ such that $\sigma \approx \sigma'$. Then, the following holds

- there exists a derivation $\delta = \sigma \xrightarrow{\omega_t^*} \sigma_1$ in P if and only if there exists a derivation $\delta' = \sigma' \xrightarrow{\omega'_t^*} \sigma'_1$ in $\text{Ann}(P)$ such $\sigma_1 \approx \sigma'_1$
- the number of **Solve (Apply)** transition steps in δ and the number of **Solve' (Apply')** transition steps in δ' are equal.

Proof

We show that any transition step from any configuration in one system can be imitated from a (possibly empty) sequence of transition steps from an equivalent configuration in the other system to achieve an equivalent configuration. Moreover there exists a **Solve (Apply)** transition step in δ if and only if there exists a **Solve' (Apply')** transition step in δ' .

Then, the proof follows by a straightforward inductive argument.

Let $\sigma = \langle (H_1, C), H_2, D, T \rangle_n \in \text{Conf}_t$ and let $\sigma' = \langle (K, C), D, T' \rangle_m \in \text{Conf}'_t$ such that $\sigma \approx \sigma'$. By definition of \approx , there exist K_1 and K_2 and a renaming ρ such that

$$K = K_1 \uplus K_2, H_1 = \text{chr}(K_1), \text{chr}(H_2) = \text{chr}(K_2), H_2\rho = K_2 \text{ and } T\rho = T'. \quad (\text{A1})$$

Solve and Solve': they move a built-in constraint from the Goal store or the Store respectively to the built-in constraint store. In this case, let $C = C' \uplus \{c\}$. By definition of the two transition systems

$$\sigma \xrightarrow{\omega_t^{\text{Solve}}} \langle (H_1, C'), H_2, D \wedge c, T \rangle_n \text{ and } \sigma' \xrightarrow{\omega'_t^{\text{Solve}'}} \langle (K, C'), D \wedge c, T' \rangle_m.$$

By definition of \approx , it is easy to check that $\langle (H_1, C'), H_2, D \wedge c, T \rangle_n \approx \langle (K, C'), D \wedge c, T' \rangle_m$.

Introduce: this kind of transition there exists only in ω_t semantics and its application labels a CHR constraint in the goal store and moves it in the CHR store. In this case, let $H_1 = H'_1 \uplus \{h\}$ and

$$\sigma \xrightarrow{\omega_t^{\text{Introduce}}} \langle (H'_1, C), H_2 \uplus \{h\#n\}, D, T \rangle_{n+1}.$$

Let $H'_2 = H_2 \uplus \{h\#n\}$. By (A1) and since $H_1 = H'_1 \uplus \{h\}$, there exists an identified atom $h\#f \in K_1$. Let $n' = \rho(n)$ (where $n' = n$ if n is not in the domain of ρ).

Now, let $K'_1 = K_1 \setminus \{h\#f\}$ and $K'_2 = K_2 \uplus \{h\#f\}$. By (A1), we have that $K = K'_1 \uplus K'_2$, $H'_1 = \text{chr}(K'_1)$ and $\text{chr}(H'_2) = \text{chr}(K'_2)$.

Moreover, by definition of \approx , for each $l \in \text{id}(K_1)$, l does not occur in T' . Therefore, since by construction $K'_1 \subseteq K_1$, we have that for each $l \in \text{id}(K'_1)$, l does not occur in T' .

Now, to prove that $\sigma' \approx \langle (H'_1, C), H'_2, D, T \rangle_{n+1}$, we have only to prove that there exists a renaming ρ' , such that $T\rho' = T'$ and $H'_2\rho' = K'_2$. We can consider the new renaming $\rho' = \rho\{n'/f, f/n'\}$. By definition ρ' is a renaming of identifiers.

Let us start proving that $H'_2\rho' = K'_2$.

We recall that $H_2\rho = K_2$ by hypothesis. Since by construction, $f \notin \text{id}(K_2) = \text{id}(H_2\rho)$, we have that $H_2\rho' = H_2\rho\{n'/f, f/n'\} = H_2\rho\{n'/f\}$. Moreover, since by definition $n \notin \text{id}(H_2)$ and $n' = \rho(n)$, we have that $H_2\rho\{n'/f\} = H_2\rho$. By the previous observations, we have that

$$H'_2\rho' = H_2\rho \uplus (\{h\#n\}\{n'/f\}) = K'_2.$$

Finally, we prove that $T\rho' = T'$. Since by definition of configurations in Conf_t , n does not occur in T and $n' = \rho(n)$, we have that $T\rho' = (T\rho)\{f/n'\} = T'\{f/n'\}$, where the last equality follows by hypothesis. Moreover since $f \in \text{id}(K_1)$, we have that f does not occur in T' . Therefore, $T'\{f/n'\} = T'$ and then the thesis.

Apply and Apply': Let $cl_r = r@F' \setminus F'' \Leftrightarrow D_1 \mid B, C_1 \in P$ and let $cl'_r = r@F' \setminus F'' \Leftrightarrow D_1 \mid \tilde{B}, C_1 \in \text{Ann}(P)$ be its annotated version, where $\tilde{B} = I(B)$. The latter can be applied to the considered configuration $\sigma' = \langle (K, C), D, T' \rangle_m$. In particular F', F'' match respectively with P_1 and P_2 such that $P_1 \uplus P_2 \subseteq K$. Without loss

of generality, by using a suitable number of **Introduce** steps, we can assume that $r@F' \setminus F'' \Leftrightarrow D_1 \mid B, C_1 \in P$ can be applied to $\sigma = \langle (H_1, C), H_2, D, T \rangle_n$. In particular, considering the hypothesis $\sigma \approx \sigma'$, we can assume for $i = 1, 2$, there exists Q_i such that $Q_1 \uplus Q_2 \subseteq H_2$, $Q_i \rho = P_i$ and F', F'' match respectively with Q_1 and Q_2 .

Then, by (A1), there exist P_3 and Q_3 such that $Q_3 \rho = P_3$, $K_2 = P_1 \uplus P_2 \uplus P_3$ and $H_2 = Q_1 \uplus Q_2 \uplus Q_3$.

By construction, since $T\rho = T'$ and $(P_1, P_2) = (Q_1, Q_2)\rho$ (and then $\text{chr}(P_1, P_2) = \text{chr}(Q_1, Q_2)$), we have that

- $r@id(P_1, P_2) \notin T'$ if and only if $r@id(Q_1, Q_2) \notin T$ and
- $\mathcal{CT} \models D \rightarrow \exists_{cl_r}(((F', F'') = \text{chr}(P_1, P_2)) \wedge D_1)$ if and only if $\mathcal{CT} \models D \rightarrow \exists_{cl_r}(((F', F'') = \text{chr}(Q_1, Q_2)) \wedge D_1)$.

Therefore, by definition of **Apply** and of **Apply'**

$$\sigma \xrightarrow{\omega_t^{Apply}} \langle \{H_1, C\} \uplus \{B, C_1\}, (Q_1, Q_3), ((F', F'') = \text{chr}(Q_1, Q_2)) \wedge D_1 \wedge D, T_1 \rangle_n$$

if and only if

$$\sigma' \xrightarrow{\omega'_t^{Apply'}} \langle (K_1, P_1, P_3, C, B', C_1), ((F', F'') = \text{chr}(P_1, P_2)) \wedge D_1 \wedge D, T'_1 \rangle_o$$

where

- $T_1 = T \cup \{r@id(Q_1, Q_2)\}$,
- $(B', \emptyset, o) = \text{inst}(\tilde{B}, \emptyset, m)$ and
- $T'_1 = T' \cup \{r@id(P_1, P_2)\}$.

Let $\sigma_1 = \langle \{H_1, C\} \uplus \{B, C_1\}, (Q_1, Q_3), ((F', F'') = \text{chr}(Q_1, Q_2)) \wedge D_1 \wedge D, T_1 \rangle_n$ and $\sigma'_1 = \langle (K_1, P_1, P_3, B', C, C_1), ((F', F'') = \text{chr}(P_1, P_2)) \wedge D_1 \wedge D, T'_1 \rangle_o$.

Now, to prove the thesis, we have to prove that $\sigma_1 \approx \sigma'_1$.

The following holds.

1. There exist $K'_1 = (K_1, B')$ and $K'_2 = (P_1, P_3)$, such that $(K_1, P_1, P_3, B') = K'_1 \cup K'_2$, $H_1 \uplus B = \text{chr}(K'_1)$ and $\text{chr}(Q_1, Q_3) = \text{chr}(K'_2)$.
2. Since for each $l \in id(K_1)$, l does not occur in T' , $P_1 \subseteq K_2$ and by definition of **Apply'** transition, we have that for each $l \in id(K'_1) = id(K_1, B')$, l does not occur in T'_1 ,
3. By construction and since $T\rho = T'$, we have that $T_1\rho = T'_1$. Moreover, by construction $(Q_1, Q_3)\rho = (P_1, P_3) = K'_2$.

By definition, we have that $\sigma_1 \approx \sigma'_1$ and then the thesis.

□

Then, we easily obtain the following

Proposition 1

Let P and $Ann(P)$ be respectively a CHR program and its annotated version. Then, for every goal G ,

$$\mathcal{QA}_P(G) = \mathcal{QA}'_{Ann(P)}(G)$$

holds.

Proof

By definition of \mathcal{QA} and of \mathcal{QA}' , the initial configurations of the two transition systems are equivalent. Then, the proof follows by Lemma 1. \square

A.2 Correctness of the unfolding

We prove now the correctness of our unfolding definition.

Next proposition states that qualified answers can be obtained by considering normal derivations only for both the semantics considered. Its proof is straightforward and hence it is omitted.

Proposition 5

Let P be CHR program and let P' an annotated CHR program. Then

$$\begin{aligned} \mathcal{QA}_P(G) = \{ \exists_{-Fv(G)}(chr(K) \wedge d) \mid & \mathcal{CT} \not\models d \leftrightarrow \mathbf{false}, \\ & \delta = \langle G, \emptyset, \mathbf{true}, \emptyset \rangle_1 \xrightarrow{\omega_t^*} \langle \emptyset, K, d, T \rangle_n \not\rightarrow_{\omega_t} \\ & \text{and } \delta \text{ is a normal derivation} \} \end{aligned}$$

and

$$\begin{aligned} \mathcal{QA}'_{P'}(G) = \{ \exists_{-Fv(G)}(chr(K) \wedge d) \mid & \mathcal{CT} \not\models d \leftrightarrow \mathbf{false}, \\ & \delta = \langle I(G), \mathbf{true}, \emptyset \rangle_m \xrightarrow{\omega_t^*} \langle K, d, T \rangle_n \not\rightarrow_{\omega_t'} \\ & \text{and } \delta \text{ is a normal derivation} \}. \end{aligned}$$

The next proposition essentially shows the correctness of unfolding w.r.t. a derivation step. We first define an equivalence between configurations in $Conf'_t$.

Definition 24 (CONFIGURATION EQUIVALENCE)

Let $\sigma = \langle G, D, T \rangle_o$ and $\sigma' = \langle G', D', T' \rangle_o$ be configurations in $Conf'_t$. σ and σ' are equivalent and we write $\sigma \simeq \sigma'$ if one of the following facts hold:

- σ and σ' are both failed configurations
- or $G = G'$, $\mathcal{CT} \models D \leftrightarrow D'$ and $clean(G, T) = clean(G', T')$.

Proposition 6

Let cl_r, cl_v be annotated CHR rules and cl'_r be the result of the unfolding of cl_r with respect to cl_v . Let σ be a generic built-in free configuration such that we can use the transition **Apply'** with the rule cl'_r obtaining the configuration $\sigma_{r'}$ and then the built-in free configuration $\sigma_{r'}^f$. Then, we can construct a derivation which uses at most the rules cl_r and cl_v and obtain a built-in free configuration σ^f such that $\sigma_{r'}^f \simeq \sigma^f$.

Proof

Assume that

$$\begin{array}{c} \sigma \xrightarrow{cl'_r} \sigma_{r'} \xrightarrow{Solve^*} \sigma_{r'}^f \\ \searrow_{cl_r} \sigma_r \xrightarrow{Solve^*} \sigma_r^f (\xrightarrow{cl_v} \sigma_v \xrightarrow{Solve^*} \sigma_v^f) \end{array}$$

The labeled arrow $\xrightarrow{Solve^*}$ means that only **Solve** transition steps are applied. Moreover:

- if σ_r^f has the form $\langle G, \mathbf{false}, T \rangle$ then the derivation between the parenthesis is not present and $\sigma^f = \sigma_r^f$.
- the derivation between the parenthesis is present and $\sigma^f = \sigma_v^f$, otherwise.

Let $\sigma = \langle (H_1, H_2, H_3), C, T \rangle_j$ be a built-in free configuration and let cl_r and cl_v be the rules $r@H'_1 \setminus H'_2 \Leftrightarrow D_r \mid K, S_1, S_2, C_r; T_r$ and $v@S'_1 \setminus S'_2 \Leftrightarrow D_v \mid P, C_v; T_v$ respectively, where C_r is the conjunction of all the built-in constraints in the body of cl_r , θ is a substitution such that $dom(\theta) \subseteq Fv(S'_1, S'_2)$ and

$$\mathcal{CT} \models (D_r \wedge C_r) \rightarrow chr(S_1, S_2) = (S'_1, S'_2)\theta. \quad (\text{A2})$$

Furthermore assume that m is the greatest identifier which appears in the rule cl_r and that $inst(P, T_v, m) = (P_1, T_1, m_1)$. Then, the *unfolded* rule cl'_r is:

$$r@H'_1 \setminus H'_2 \Leftrightarrow D_r, (D'_v\theta) \mid K, S_1, P_1, C_r, C_v, chr(S_1, S_2) = (S'_1, S'_2); T_{r'}$$

where $v@id(S_1, S_2) \notin T_r$, $V \subseteq D_v$, $V = \{c \mid \mathcal{CT} \models (D_r \wedge C_r) \rightarrow c\theta\}$, $D'_v = D_v \setminus V$, $Fv(D'_v\theta) \cap Fv((S'_1, S'_2)\theta) \subseteq Fv(H'_1, H'_2)$, the constraint $(D_r, (D'_v\theta))$ is satisfiable and

- if $S'_2 = \epsilon$ then $T_{r'} = T_r \cup T_1 \cup \{v@id(S_1)\}$
- if $S'_2 \neq \epsilon$ then $T_{r'} = clean((K, S_1), T_r) \cup T_1$.

By the previous observations, we have that

$$\mathcal{CT} \models (D_r \wedge C_r) \rightarrow V\theta, \quad (\text{A3})$$

and therefore $\mathcal{CT} \models V\theta \leftrightarrow \exists_{-Fv(D_r \wedge C_r)} V\theta$. Then, without loss of generality, we can assume that

$$Fv(V\theta) \subseteq Fv(cl_r). \quad (\text{A4})$$

Analogously, by (A2) and since $dom(\theta) \subseteq Fv(S'_1, S'_2)$, we can assume that

$$Fv(chr(S_1, S_2) = (S'_1, S'_2)\theta) = Fv((chr(S_1, S_2) = (S'_1, S'_2))\theta) \subseteq Fv(cl_r). \quad (\text{A5})$$

Moreover, since by definition $Fv(D'_v\theta) \cap Fv((S'_1, S'_2)\theta) \subseteq Fv(H'_1, H'_2)$ and $dom(\theta) \subseteq Fv(S'_1, S'_2)$, we have that

$$Fv(D'_v\theta) \subseteq Fv(H'_1, H'_2) \cup Fv(cl_v). \quad (\text{A6})$$

Let us consider the application of the rule cl'_r to σ . By definition of the transition **Apply'**, we have that

$$\mathcal{CT} \models C \rightarrow \exists_{cl'_r} ((chr(H_1, H_2) = (H'_1, H'_2)) \wedge D_r \wedge (D'_v\theta)) \quad (\text{A7})$$

and

$$\sigma_{r'} = \langle (Q, C_r, C_v, \text{chr}(S_1, S_2) = (S'_1, S'_2)), D, T_3 \rangle_{j+m_1},$$

where

- $Q = (H_1, H_3, Q_1)$,
- $\mathcal{CT} \models D \leftrightarrow (\text{chr}(H_1, H_2) = (H'_1, H'_2) \wedge D_r \wedge (D'_v \theta) \wedge C)$,
- $\text{inst}((K, S_1, P_1), T_{r'}, j) = (Q_1, T'_{r'}, j + m_1)$ and $T_3 = T \cup T'_{r'} \cup \{r@id(H_1, H_2)\}$.

Therefore, by definition $\sigma_{r'}^f = \langle Q, C_{r'}^f, T_3 \rangle_{j+m_1}$, where

$$\mathcal{CT} \models C_{r'}^f \leftrightarrow (C_r \wedge C_v \wedge \text{chr}(S_1, S_2) = (S'_1, S'_2) \wedge D).$$

Let us consider now the application of cl_r to σ and then of cl_v to the σ_r^f obtained from the previous application. Since by construction $Fv((\text{chr}(H_1, H_2) = (H'_1, H'_2)) \wedge D_r) \cap Fv(cl'_r) \subseteq Fv(cl_r)$ and by (A7), we have that

$$\mathcal{CT} \models C \rightarrow \exists_{cl_r}((\text{chr}(H_1, H_2) = (H'_1, H'_2)) \wedge D_r).$$

Therefore, by definition of the transition **Apply'**, we have that

$$\sigma_r = \langle (Q_2, C_r), \text{chr}(H_1, H_2) = (H'_1, H'_2) \wedge D_r \wedge C, T_4 \rangle_{j+m},$$

where

- $Q_2 = (H_1, H_3, K'', S''_1, S''_2)$,
- $((K'', S''_1, S''_2), T_2, j+m) = \text{inst}((K, S_1, S_2), T_r, j)$ and $T_4 = T \cup T_2 \cup \{r@id(H_1, H_2)\}$.

Therefore, by definition $\sigma_r^f = \langle Q_2, C_r^f, T_4 \rangle_{j+m}$, where

$$\mathcal{CT} \models C_r^f \leftrightarrow C_r \wedge \text{chr}(H_1, H_2) = (H'_1, H'_2) \wedge D_r \wedge C. \quad (\text{A8})$$

Now, we have two possibilities

($C_r^f = \text{false}$). In this case, by construction, we have that $C_{r'}^f = \text{false}$. Therefore $\sigma_{r'}^f \simeq \sigma_r^f$ and then the thesis.

($C_r^f \neq \text{false}$). By (A8) and (A3) (A2), we have that

$$\mathcal{CT} \models C_r^f \rightarrow \text{chr}(S_1, S_2) = (S'_1, S'_2) \theta \wedge V \theta.$$

Moreover, by (A8), (A7) and (A6)

$$\begin{aligned} \mathcal{CT} \models C_r^f \rightarrow & \exists_{H'_1, H'_2, cl_v} (\text{chr}(H_1, H_2) = (H'_1, H'_2) \wedge (D'_v \theta)) \\ & \wedge \text{chr}(H_1, H_2) = (H'_1, H'_2) \end{aligned}$$

and then $\mathcal{CT} \models C_r^f \rightarrow \exists_{cl_v} (D'_v \theta)$. Therefore, by (A4), (A5) and since the rules are renamed apart,

$$\mathcal{CT} \models C_r^f \rightarrow \exists_{cl_v} (\text{chr}(S_1, S_2) = (S'_1, S'_2) \theta \wedge V \theta \wedge D'_v \theta).$$

Then, by definition of D_v and since $\text{dom}(\theta) \subseteq Fv(S'_1, S'_2)$, we have that $\mathcal{CT} \models C_r^f \rightarrow \exists_{cl_v} ((\text{chr}(S_1, S_2) = (S'_1, S'_2) \wedge D_v) \theta)$.

Therefore, since $\text{dom}(\theta) \subseteq Fv(S'_1, S'_2) \subseteq Fv(cl_v)$,

$$\mathcal{CT} \models C_r^f \rightarrow \exists_{cl_v} (\text{chr}(S_1, S_2) = (S'_1, S'_2) \wedge D_v).$$

Then, σ_r^f is such that we can use the transition **Apply'** with the rule cl_v obtaining the new configuration

$$\sigma_v = \langle (Q_3, C_v), D', T_5 \rangle_{m_1},$$

where

- $Q_3 = (H_1, H_3, K'', S_1'', P_2)$
- $\mathcal{CT} \models D' \leftrightarrow (chr(S_1, S_2) = (S_1', S_2') \wedge D_v \wedge C_r \wedge chr(H_1, H_2) = (H_1', H_2') \wedge D_r \wedge C)$,
- $inst(P, T_v, j + m) = (P_2, T'_v, m_1)$ and $T_5 = T_4 \cup T'_v \cup \{v@id(S_1'', S_2'')\}$.

Finally, by definition, we have that $\sigma_v^f = \langle Q_3, C_v^f, T_5 \rangle_{m_1}$, where

$$\mathcal{CT} \models C_v^f \leftrightarrow C_v \wedge D'.$$

By definition of D and D' , we have that $\mathcal{CT} \models C_{r'}^f \leftrightarrow C_v^f$.

If $C_v^f = \mathbf{false}$ then the proof is analogous to the previous case and hence it is omitted. Otherwise, observe that by construction, $Q = (H_1, H_3, Q_1)$, where Q_1 is obtained from (K, S_1, P_1) by adding the natural j to each identifier in (K, S_1) and by adding the natural $j + m$ to each identifier in P . Analogously, by construction, $Q_3 = (H_1, H_3, K'', S_1'', P_2)$, where (K'', S_1'') are obtained from (K, S_1) by adding the natural j to each identifier in (K, S_1) and P_2 is obtained from P by adding the natural $j + m$ to each identifier in P .

Therefore $Q = Q_3$ and then, to prove the thesis, we have only to prove that

$$clean(Q, T_3) = clean(Q, T_5).$$

Let us introduce the function $inst' : Token \times \mathbb{N} \rightarrow \mathbb{N}$ as the restriction of the function $inst$ to token sets and natural numbers, namely $inst'(T, n) = T'$, where T' is obtained from T by incrementing each identifier in T with n . So, since $T_{r'} = inst'(T_{r'}, j)$, $T_{r'} = T_r \cup T_1 \cup \{v@id(S_1, S_2)\}$ and $T_1 = inst'(T_v, m)$, we have that

$$\begin{aligned} T_3 &= T \cup T_{r'} \cup \{r@id(H_1, H_2)\} \\ &= T \cup inst'(clean((K, S_1), T_r), j) \cup inst'(T_v, j + m) \cup \\ &\quad inst'(\{v@id(S_1, S_2)\}, j) \cup \{r@id(H_1, H_2)\} \end{aligned}$$

Analogously, since $T_4 = T \cup T_2 \cup \{r@id(H_1, H_2)\}$, $T_2 = inst'(T_r, j)$ and $T'_v = inst'(T_v, j + m)$, we have that

$$\begin{aligned} T_5 &= T_4 \cup T'_v \cup \{v@id(S_1'', S_2'')\} \\ &= T \cup inst'(T_r, j) \cup \{r@id(H_1, H_2)\} \cup inst'(T_v, j + m) \cup \\ &\quad \{v@id(S_1'', S_2'')\} \end{aligned}$$

Now, since by construction (S_1'', S_2'') is obtained from (S_1, S_2) by adding the natural j to each identifier, we have that $inst'(\{v@id(S_1, S_2)\}, j) = \{v@id(S_1'', S_2'')\}$. Moreover, by definition of annotated rule $id(T_r) \subseteq id(K, S_1, S_2)$ and $Q = (H_1, H_3, Q_1)$, where Q_1 is obtained from (K, S_1, P_1) by adding the natural j to each identifier in (K, S_1) and by adding the natural $j + m$ to each identifier in P . Then

$clean(Q, inst'(clean((K, S_1), T_r), j)) = clean(Q, inst'(T_r, j))$ and then the thesis holds.

□

Hence we obtain the correctness result.

Proposition 2

Let P be an annotated CHR program with $cl_r, cl_v \in P$. Let cl'_r be the result of the unfolding of cl_r with respect to cl_v and let P' be the program obtained from P by adding rule cl'_r . Then, for every goal G , $\mathcal{QA}'_{P'}(G) = \mathcal{QA}'_P(G)$ holds.

Proof

We prove the two inclusions separately.

$(\mathcal{QA}'_{P'}(G) \subseteq \mathcal{QA}'_P(G))$ The proof follows from Propositions 5 and 6 and by a straightforward inductive argument.

$(\mathcal{QA}'_P(G) \subseteq \mathcal{QA}'_{P'}(G))$ The proof is by contradiction. Assume that there exists $Q \in \mathcal{QA}'_P(G) \setminus \mathcal{QA}'_{P'}(G)$. By definition there exists a derivation

$$\delta = \langle I(G), \mathbf{true}, \emptyset \rangle_m \rightarrow_{\omega'_t}^* \langle K, d, T \rangle_n \not\rightarrow_{\omega'_t}$$

in P , such that $Q = \exists_{-Fv(G)}(chr(K) \wedge d)$. Since $P \subseteq P'$, we have that there exists the derivation $\langle I(G), \mathbf{true}, \emptyset \rangle_m \rightarrow_{\omega'_t}^* \langle K, d, T \rangle_n$ in P' . Moreover, since $P' = P \cup \{cl'_r\}$ and by hypothesis $Q \notin \mathcal{QA}'_{P'}(G)$, we have that there exists a derivation step $\langle K, d, T \rangle_n \rightarrow_{\omega'_t} \langle K_1, d_1, T_1 \rangle_{n_1}$ by using the rule cl'_r . Then, by definition of unfolding there exists a derivation step $\langle K, d, T \rangle_n \rightarrow_{\omega'_t} \langle K_2, d_2, T_2 \rangle_{n_2}$ in P , by using the rule cl_r and then we have a contradiction.

□

A.3 Safe replacement

We can now provide the result which shows the correctness of the safe rule replacement condition. This is done by using the following proposition.

Proposition 7

Let cl_r, cl_v be two annotated CHR rules such that the following holds

- cl_r is of the form $r @ H'_1 \setminus H'_2 \Leftrightarrow D_r \mid K_r; T_r$,
- $cl'_r \in Unf_{\{cl_v\}}(cl)$ is of the form $r @ H'_1 \setminus H'_2 \Leftrightarrow D'_r \mid K'_r; T'_r$, with $\mathcal{CT} \models D_r \leftrightarrow D'_r$ and it is obtained by unfolding the identified atoms $A \subseteq K_r$.

Moreover, let σ be a generic built-in free configuration such that we can construct a derivation δ from σ where

- δ uses at the most the rules cl_r and cl_v in the order,
- a built-in free configuration σ^f can be obtained and
- if cl_v is used, then cl_v rewrites the atoms A' such that $chr(A) = chr(A')$.

Then, we can use the transition **Apply'** with the rule cl'_r obtaining the configuration $\sigma_{r'}$ and then the built-in free configuration $\sigma_{r'}^f$ such that $\sigma_{r'}^f \simeq \sigma^f$.

Proof

Assume that

$$\begin{array}{c} \sigma \longrightarrow^{cl_r} \sigma_r \longrightarrow^{Solve^*} \sigma_r^f \longrightarrow^{cl_v} \sigma_v \longrightarrow^{Solve^*} \sigma_v^f \\ \searrow^{cl'_r} \sigma_{r'} \longrightarrow^{Solve^*} \sigma_{r'}^f \end{array}$$

The labeled arrow $\longrightarrow^{Solve^*}$ means that only **Solve** transition steps are applied.

Moreover

- if σ_r^f has the form $\langle G, \mathbf{false}, T \rangle$ then the derivation between the parenthesis is not present and $\sigma^f = \sigma_r^f$.
- the derivation between the parenthesis is present and $\sigma^f = \sigma_v^f$, otherwise.

We first need some notation. Let $\sigma = \langle (F_1, F_2, F_3), C, T \rangle_j$ be a built-in free configuration and let cl_r and cl_v be of the form $r@H_1 \setminus H_2 \Leftrightarrow D_r \mid K, A, C_r; T_r$ and $v@H'_1 \setminus H'_2 \Leftrightarrow D_v \mid P, C_v; T_v$ respectively, $A = A_1 \uplus A_2$, C_r is the conjunction of all the built-in constraints in the body of cl_r and θ is a substitution such that $dom(\theta) \subseteq Fv(H'_1, H'_2)$ and

$$\mathcal{CT} \models (D_r \wedge C_r) \rightarrow chr(A_1, A_2) = (H'_1, H'_2)\theta. \quad (\text{A9})$$

Furthermore let m be the greatest identifier which appears in the rule cl_r and let $(P_1, T_1, m_1) = inst(P, T_v, m)$.

Then, the *unfolded* rule cl'_r is:

$$r@H_1 \setminus H_2 \Leftrightarrow D_r, (D'_v\theta) \mid K, A_1, P_1, C_r, C_v, chr(A_1, A_2) = (H'_1, H'_2); T'_r$$

where $v@id(A_1, A_2) \notin T_r$, $V = \{d \in D_v \mid \mathcal{CT} \models (D_r \wedge C_r) \rightarrow d\theta\}$, $D'_v = D_v \setminus V$, $Fv(D'_v\theta) \cap Fv(k'\theta) \subseteq Fv(H_1, H_2)$, the constraint $(D_r, (D'_v\theta))$ is satisfiable and

- if $H'_2 = \epsilon$ then $T'_r = T_r \cup T_1 \cup \{v@id(A_1)\}$
- if $H'_2 \neq \epsilon$ then $T'_r = clean((K, A_1), T_r) \cup T_1$.

Since by hypothesis, $\mathcal{CT} \models (D_r, (D'_v\theta)) \leftrightarrow D_r$, we have that

$$\mathcal{CT} \models (D_r \wedge C_r) \rightarrow D_v\theta \text{ and } D'_v\theta = \emptyset. \quad (\text{A10})$$

Let us now consider the application of the rule cl_r to σ . By definition of the **Apply'** transition step, we have that

$$\mathcal{CT} \models C \rightarrow \exists_{cl_r}. ((chr(F_1, F_2) = (H_1, H_2)) \wedge D_r) \quad (\text{A11})$$

and

$$\sigma_r = \langle (Q_2, C_r), chr(F_1, F_2) = (H_1, H_2) \wedge D_r \wedge C, T_4 \rangle_{j+m},$$

where $Q_2 = (F_1, F_3, K', A')$, $((K', A'), T_2, j + m) = inst((K, A), T_r, j)$ and $T_4 = T \cup T_2 \cup \{r@id(F_1, F_2)\}$.

Therefore, by definition

$$\sigma_r^f = \langle Q_2, C_r^f, T_4 \rangle_{j+m}.$$

where

$$\mathcal{CT} \models C_r^f \leftrightarrow C_r \wedge chr(F_1, F_2) = (H_1, H_2) \wedge D_r \wedge C. \quad (\text{A12})$$

Let us now apply the rule cl'_r to σ . By (A11), (A10) and by definition of the **Apply**' transition step, we have that

$$\sigma_{r'} = \langle (Q, C_r, C_v, chr(A_1, A_2) = (H'_1, H'_2)), D, T_3 \rangle_{j+m_1},$$

where

- $\mathcal{CT} \models D \leftrightarrow chr(F_1, F_2) = (H_1, H_2) \wedge D_r \wedge C$,
- $Q = (F_1, F_3, Q_1)$,
- $inst((K, A_1, P_1), T'_r, j) = (Q_1, T''_r, j + m_1)$ and $T_3 = T \cup T''_r \cup \{r@id(F_1, F_2)\}$.

Therefore, by definition

$$\sigma_{r'}^f = \langle Q, C_{r'}^f, T_3 \rangle_{j+m_1}.$$

where

$$\mathcal{CT} \models C_{r'}^f \leftrightarrow C_r \wedge C_v \wedge chr(A_1, A_2) = (H'_1, H'_2) \wedge D.$$

Now, we consider the two previously obtained configurations σ_r^f and $\sigma_{r'}^f$. Since by hypothesis σ_v^f is a non-failed configuration, we have that $C_r^f \neq \mathbf{false}$

Now, let $A' \in Q_2$ such that $chr(A') = chr(A)$. Note that such atoms there exist, since by construction A are atoms in the body of cl_r .

By definition, since A are atoms in the body of cl_r , $dom(\theta) \subseteq Fv(H'_1, H'_2) \subseteq Fv(cl_v)$, by (A12), (A9) and (A10), we have that

$$\mathcal{CT} \models C_r^f \rightarrow ((chr(A_1, A_2) = (H'_1, H'_2)) \wedge D_v)\theta$$

and therefore, since $dom(\theta) \subseteq Fv(cl_v)$, we have that

$$\mathcal{CT} \models C_r^f \rightarrow \exists_{cl_v}((chr(A_1, A_2) = (H'_1, H'_2)) \wedge D_v).$$

Then, since by hypothesis cl_v rewrites the atom $A = (A_1, A_2)$ such that $chr(A') = chr(A'_1, A'_2) = chr(A_1, A_2) = chr(A)$, we have that

$$\sigma_v = \langle (Q_3, C_v), D', T_5 \rangle_{m_1},$$

where

- $Q_3 = (F_1, F_3, K', A'_1, P_2)$,
- $D' = (chr(A_1, A_2) = (H'_1, H'_2)) \wedge D_v \wedge C_r \wedge chr(F_1, F_2) = (H_1, H_2) \wedge D_r \wedge C$,
- $inst(P, T_v, j + m) = (P_2, T'_v, m_1)$ and $T_5 = T_4 \cup T'_v \cup \{v@id(k'')\}$.

Finally, by definition, we have that $\sigma_v^f = \langle Q_3, C_v^f, T_5 \rangle_{m_1}$, where

$$\mathcal{CT} \models C_v^f \leftrightarrow (C_v \wedge D').$$

If $C_v^f = \mathbf{false}$ then the proof is analogous to the previous case and hence it is omitted.

Otherwise, the proof is analogous to that given for Proposition 6 and hence it is omitted. \square

Proposition 8

Let $\sigma_0 = \langle F, c, T \rangle_m$ be a built-in configuration and let cl be an annotated CHR rule such that the following holds.

- a) $cl = r@H_1 \setminus H_2 \Leftrightarrow D \mid A; T$. where $(H_1, H_2) = (h_1, \dots, h_n)$,
- b) there exists $(K_1, K_2) = (k_1, \dots, k_n) \subseteq F$ such that $r@id(K_1, K_2) \notin T$ and $\mathcal{CT} \models c \rightarrow \exists_{cl}((chr(K_1, K_2) = (H_1, H_2)) \wedge D)$,
- c) there exist $l \in \{1, \dots, n\}$ and $(K'_1, K'_2) = (k'_1, \dots, k'_n) \subseteq F$ such that $k_l = k'_l$, $r@id(K'_1, K'_2) \notin T$ and $\mathcal{CT} \models c \wedge chr(k_l) = h_l \rightarrow (chr(K'_1, K'_2) = (H_1, H_2))$,
- d) $\sigma_0 \xrightarrow{\omega'_t} \sigma$ is an **Apply'** transition step which uses the clause cl , rewrites the atoms (K_1, K_2) and such that $\sigma = \langle (F \setminus K_1) \uplus A', C, T' \rangle_{m'}$, where C is the constraint $(chr(K_1, K_2) = (H_1, H_2)) \wedge D \wedge c$.

Then, there exists an **Apply'** transition step $\sigma_0 \xrightarrow{\omega'_t} \sigma'$ which uses the clause cl , rewrites the atoms (K'_1, K'_2) and such that $\sigma' = \langle (F \setminus K'_1) \uplus A', C', T'' \rangle_{m'}$, where C' is the constraint $(chr(K'_1, K'_2) = (H_1, H_2)) \wedge D \wedge c$ and

1. $\mathcal{CT} \models ((chr(F \setminus K_1) \wedge A') \wedge C) \Leftrightarrow ((chr(F \setminus K'_1) \wedge A') \wedge C')$,
2. $T'' = (T' \setminus \{r@id(K_1, K_2)\}) \cup r@id(K'_1, K'_2)$.

Proof

First of all, by definition of **Apply'** transition step and since, by hypothesis **c)**, $r@id(K'_1, K'_2) \notin T$, we have to prove that

$$\mathcal{CT} \models c \rightarrow \exists_{cl}((chr(K'_1, K'_2) = (H_1, H_2)) \wedge D).$$

By hypothesis **b)** and since $Fv(c) \cap Fv(cl) = \emptyset$, we have that $\mathcal{CT} \models c \rightarrow \exists_{cl}(c \wedge (chr(k_l) = h_l) \wedge D)$. Hence the thesis follows by hypothesis **c)**.

Now, we have to prove 1. By hypothesis **b)**, we have that $\mathcal{CT} \models c \rightarrow \exists_{cl}(chr(K_1, K_2) = (H_1, H_2))$. Therefore, there exists a substitution ϑ such that $dom(\vartheta) = Fv(H_1, H_2)$ and

$$\mathcal{CT} \models c \rightarrow (chr(K_1, K_2) = (H_1, H_2)\vartheta). \quad (\text{A13})$$

By hypothesis **c)** and since $dom(\vartheta) \cap Fv(c, K_1, K_2) = \emptyset$, we have that

$$\mathcal{CT} \models (c \wedge (chr(k_l) = h_l\vartheta)) \rightarrow (chr(K'_1, K'_2) = (H_1, H_2)\vartheta)$$

and by (A13), $\mathcal{CT} \models c \rightarrow (chr(k_l) = (h_l)\vartheta)$.

Then $\mathcal{CT} \models c \rightarrow (chr(K'_1, K'_2) = chr(K_1, K_2))$ and then the thesis.

The proof of 2 is obvious by definition of **Apply'** transition step. \square

Proposition 9

Let $\sigma_0 = \langle F, c, T \rangle_m$ be a built-in configuration such that there exists a normal terminating derivation δ starting from σ which ends in a configuration σ . Assume that δ uses an annotated CHR rule cl such that the following holds.

- a) $cl = r@H_1 \setminus H_2 \Leftrightarrow D \mid A; T$
- b) there exists $(K_1, K_2) \subseteq F$ such that cl rewrites the atoms (K_1, K_2) in δ and $\mathcal{CT} \models c \rightarrow \exists_{cl}((chr(K_1, K_2) = (H_1, H_2)) \wedge D)$

Then, there exists a normal terminating derivation δ' starting from σ_0 such that

- δ' uses at most the same clauses of δ and uses the rule cl in the first **Apply'** transition step, in order to rewrite the atoms (K_1, K_2) ,
- δ' ends in a configuration σ' such that $\sigma \simeq \sigma'$.

Proof

The proof is obvious by definition of derivation. \square

Hence we have the following result.

Theorem 1

Let P be an annotated program, cl be a rule in P such that cl can be safely replaced in P according to Definition 11. Assume also that

$$P' = (P \setminus \{cl\}) \cup Unf_P(cl).$$

Then $\mathcal{QA}'_P(G) = \mathcal{QA}'_{P'}(G)$ for any arbitrary goal G .

Proof

By using a straightforward inductive argument and by Proposition 2, we have that $\mathcal{QA}'_P(G) = \mathcal{QA}'_{P''}(G)$ where

$$P'' = P \cup Unf_P(cl),$$

for any arbitrary goal G .

Then, to prove the thesis, we have only to prove that

$$\mathcal{QA}'_{P'}(G) = \mathcal{QA}'_{P''}(G).$$

In the following, we assume that cl is of the form $r@H_1 \setminus H_2 \Leftrightarrow D \mid A; T$. We prove the two inclusions separately.

$(\mathcal{QA}'_{P'}(G) \subseteq \mathcal{QA}'_{P''}(G))$ The proof is by contradiction. Assume that there exists $Q \in \mathcal{QA}'_{P'}(G) \setminus \mathcal{QA}'_{P''}(G)$. By definition there exists a derivation

$$\delta = \langle I(G), \mathbf{true}, \emptyset \rangle_m \rightarrow_{\omega'_t}^* \langle K, d, T \rangle_n \not\rightarrow_{\omega'_t}$$

in P' , such that $Q = \exists_{-Fv(G)}(chr(K) \wedge d)$. Since $P' \subseteq P''$, we have that there exists the derivation

$$\langle I(G), \mathbf{true}, \emptyset \rangle_m \rightarrow_{\omega'_t}^* \langle K, d, T \rangle_n$$

in P'' . Moreover, since $P'' = P' \cup \{cl\}$ and $Q \notin \mathcal{QA}'_{P''}(G)$, we have that there exists a derivation step $\langle K, d, T \rangle_n \rightarrow_{\omega'_t} \langle K_1, d_1, T_1 \rangle_{n_1}$ by using the rule cl .

Since cl can be safely replaced in P , we have that there exists an unfolded rule $cl' \in Unf_P(cl)$ such that cl' is of the form

$$r@H_1 \setminus H_2 \Leftrightarrow D' \mid A'; T',$$

$\mathcal{CT} \models D \Leftrightarrow D'$ and by construction $cl' \in P'$.

Then, there exists a derivation step $\langle K, d, T \rangle_n \rightarrow_{\omega'_t} \langle K_2, d_2, T_2 \rangle_{n_2}$ in P' (by using the rule cl') and then we have a contradiction.

$(\mathcal{QA}'_{P''}(G) \subseteq \mathcal{QA}'_{P'}(G))$ First of all, observe that by Proposition 5, $\mathcal{QA}'_{P''}(G)$ can be calculated by considering only non-failed normal terminating derivations. Then, for each non-failed normal terminating derivation δ in P'' , which uses the rule cl after the application of cl , we obtain the configuration σ_1 and then a non-failed built-in free configuration σ_1^f . Now, let C be the built-in constraint store of σ_1^f .

Since by hypothesis cl can be safely replaced in P , following Definition 11, we have there exists at least an atom $k \in A$, such that there exists a corresponding atom (in the obvious sense) k' which is rewritten in δ by using a rule cl' in P . Therefore, without loss of generality, we can assume that

$$\delta = \langle I(G), \mathbf{true}, \emptyset \rangle_m \rightarrow_{\omega'_t}^* \sigma \rightarrow_{\omega'_t} \sigma_1 \rightarrow_{\omega'_t}^* \sigma_1^f \rightarrow_{\omega'_t}^* \sigma_2 \rightarrow_{\omega'_t} \sigma_3 \rightarrow_{\omega'_t}^* \sigma_4 \not\rightarrow_{\omega'_t}^*$$

where the transition step $s_1 = \sigma \rightarrow_{\omega'_t} \sigma_1$ is the first **Apply'** transition step which uses the clause cl and $s_2 = \sigma_2 \rightarrow_{\omega'_t} \sigma_3$ is the first **Apply'** transition step which rewrites an atom k' , corresponding to an atom k in the body of cl introduced by s_1 . Since by hypothesis cl can be safely replaced in P and by Proposition 8 we can assume that cl' rewrites in s_2 only atoms corresponding (in the obvious sense) to atoms in A . Moreover, since by hypothesis cl can be safely replaced in P and by Proposition 9, we can assume that s_2 is the first **Apply'** transition step after s_1 . Then, the thesis follows since by hypothesis cl can be safely replaced in P , by Proposition 7 and by a straightforward inductive argument.

□

A.4 Termination and confluence

We first prove the correctness of unfolding w.r.t. termination.

Proposition 3 (NORMAL TERMINATION)

Let P be a CHR program and let P_0, \dots, P_n be an U-sequence starting from $Ann(P)$. P satisfies normal termination if and only if P_n satisfies normal termination.

Proof

By Lemma 1, we have that P is normally terminating if and only if $Ann(P)$ is normally terminating. Moreover from Proposition 6 and Proposition 7 and by using a straightforward inductive argument, we have that for each $i = 0, \dots, n-1$, P_i satisfies normal termination if and only if P_{i+1} satisfies the normal termination too and then the thesis. □

The following lemma relates the \approx , \simeq and \equiv_V equivalences.

Lemma 2

Let σ, σ' be final configurations in $Conf_t$, $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2 \in Conf'_t$ and let V be a set of variables.

- If $\sigma_1 \approx \sigma$, $\sigma'_1 \approx \sigma'$ then $\sigma_1 \equiv_V \sigma'_1$ if and only if $\sigma \equiv_V \sigma'$.
- If $\sigma_1 \simeq \sigma_2$, $\sigma'_1 \simeq \sigma'_2$ and $\sigma_1 \equiv_V \sigma'_1$ then $\sigma_2 \equiv_V \sigma'_2$.

Proof

The proof of the first statement follows by definition of \approx and by observing that if σ is a final configuration in $Conf_t$, then σ has the form $\langle G, S, \mathbf{false}, T \rangle_n$ or it has the form $\langle \emptyset, S, c, T \rangle_n$.

The proof of the second statement is straightforward, by observing that if $\sigma_1 \simeq \sigma_2$, then $\sigma_1 \equiv_V \sigma_2$ for each set of variables V . \square

Theorem 2 (CONFLUENCE)

Let P be a CHR program and let P_0, \dots, P_n be an NRU-sequence starting from $P_0 = Ann(P)$. P satisfies confluence if and only if P_n satisfies confluence too.

Proof

By Lemma 1, we have that P is confluent if and only if $Ann(P)$ is confluent. Moreover, by Proposition 6 Now, we prove that for each $i = 0, \dots, n-1$, P_i is confluent if and only if $P_{i+1} = (P_i \setminus \{cl^i\}) \cup Unf_{P_i}(cl^i)$ is confluent too. Then, the proof follows by a straightforward inductive argument.

- Assume that P_i is confluent and let us assume by contrary that P_{i+1} does not satisfies confluence. By definition, there exists a state $\sigma = \langle (K, D), C, T \rangle_o$ and two derivations $\sigma \rightarrow_{\omega'_t}^* \sigma_1$ and $\sigma \rightarrow_{\omega'_t}^* \sigma_2$ in P_{i+1} such that there are no two derivations $\sigma_1 \mapsto^* \sigma'_1$ and $\sigma_2 \mapsto^* \sigma'_2$ in P_{i+1} where $\sigma'_1 \equiv_{Fv(\sigma)} \sigma'_2$. Without loss of generality, we can assume that σ_1 and σ_2 are built-in free states. Therefore, by Proposition 6, there exist two derivations $\sigma \rightarrow_{\omega'_t}^* \sigma_3$ and $\sigma \rightarrow_{\omega'_t}^* \sigma_4$ in P_i , such that $\sigma_1 \simeq \sigma_3$ and $\sigma_2 \simeq \sigma_4$. Moreover, since P_i is confluent, there exist two derivations $\delta = \sigma_3 \rightarrow_{\omega'_t}^* \sigma'_3$ and $\delta' = \sigma_4 \rightarrow_{\omega'_t}^* \sigma'_4$ in P_i such that $\sigma'_3 \equiv_{Fv(\sigma)} \sigma'_4$. Moreover, without loss of generality, we can assume that σ'_3 and σ'_4 are built-in free. Analogously to Theorem 1, since by hypothesis cl^i can be safely replaced in P_i and by using Proposition 7, we can construct two new derivations $\gamma = \sigma_1 \rightarrow_{\omega'_t}^* \sigma_5$ and $\gamma' = \sigma_2 \rightarrow_{\omega'_t}^* \sigma_6$ in $P_i \cup Unf_{P_i}(cl^i)$ such that σ_5 and σ_6 are built-in free, $\sigma_5 \simeq \sigma'_3$, $\sigma_6 \simeq \sigma'_4$ and such that if γ and γ' use the clause cl^i , then no atoms introduced (in the obvious sense) by cl_i is rewritten by using (at least) one rule in $P_i \cup Unf_{P_i}(cl^i)$. Moreover, by hypothesis and by Lemma 2, $\sigma_5 \equiv_{Fv(\sigma)} \sigma_6$.

Let l the number of the **Apply**' transition steps in δ and δ' , which use the rule cl^i and whose body is not rewritten by using (at least) one rule in $P_i \cup Unf_{P_i}(cl^i)$. The proof is by induction on l .

($l = 0$) In this case, $\gamma = \sigma_1 \rightarrow_{\omega'_t}^* \sigma_5$ and $\gamma' = \sigma_2 \rightarrow_{\omega'_t}^* \sigma_6$ are derivations in P_{i+1} .

By hypothesis and by Lemma 2, we have that $\sigma_5 \equiv_{Fv(\sigma)} \sigma_6$ and then we have a contradiction.

($l > 0$) Let us consider the last **Apply**' transition step in γ and γ' , which use (a renamed version of) the rule $cl^i = r_i @ H_1 \setminus H_2 \Leftrightarrow D \mid K, C; T$, whose body is not rewritten by using (at least) one rule in $P_i \cup Unf_{P_i}(cl^i)$ and where C is the conjunction of all the built-in constraints in the body of cl^i . Without loss of

generality, we can assume that such an **Apply**' transition step is in γ . Now, we have two possibilities

- σ_5 is a failed configuration. By definition of $\equiv_{Fv(\sigma)}$, we have that σ is also a failed configuration. In this case, it is easy to check that, by using Lemma 7, we can substitute each **Apply**' transition steps in δ and δ' , which use the rule cl^i and whose body is not rewritten by using (at least) one rule P_i , with an **Apply**' transition step which uses a rule in $Unf_{P_i}(cl^i) \subseteq P_{i+1}$. Then, analogously to the case ($l = 0$), it is easy to check that there exist the derivations $\gamma_1 = \sigma_1 \rightarrow_{\omega'_t}^* \sigma'_5$ and $\gamma'_1 = \sigma_2 \rightarrow_{\omega'_t}^* \sigma'_6$ in P_{i+1} such that σ'_3 and σ'_4 are both failed configurations and then we have a contradiction.
- σ_5 is not a failed configuration. Then σ_5 is of the form $\langle S_5, C_5, T_5 \rangle_{n_5}$, where $chr(K) \subseteq chr(S_5)$. Moreover, since cl^i can be non-recursively safely replaced in P_i , there exists a clause cl_v in $P_i \setminus \{cl^i\}$ such that cl^i can be unfolded by using cl_v . Therefore, by definition of non-recursive safe unfolding, there exists a new derivation $\gamma_1 = \sigma_1 \rightarrow_{\omega'_t}^* \sigma_5 \rightarrow_{\omega'_t}^* \sigma'_5$, where σ'_5 is obtained from σ_5 first by an **Apply**' transition step, which uses the rule cl_v and rewrites atoms in the body of cl^i and then some **Solve**' transition steps. By definition of $\equiv_{Fv(\sigma)}$ and since $\sigma_5 \equiv_{Fv(\sigma)} \sigma_6$, we have that there exists also a new derivation $\gamma'_1 = \sigma_2 \rightarrow_{\omega'_t}^* \sigma_6 \rightarrow_{\omega'_t}^* \sigma'_6$, where σ'_6 is obtained from σ_6 first by an **Apply**' transition step, which uses the rule cl_v and rewrites atoms in the body of cl^i and then some **Solve**' transition steps.

Since by hypothesis $\sigma_5 \equiv_{Fv(\sigma)} \sigma_6$, we have that $\sigma'_5 \equiv_{Fv(\sigma)} \sigma'_6$. Moreover the number of the **Apply**' transition steps in δ_2 and δ'_2 , which use the rule cl^i whose body is not rewritten by using (at least) one rule in P_i is strictly less than l and then the thesis.

- Assume that P_{i+1} is confluent and let us assume by contrary that P_i does not satisfies confluence. The proof is analogous to the previous case and hence it is omitted.

□

A.5 Weak safe rule replacement

Finally, we provide the proof of Proposition 4. We first need of the following lemma, which provides an alternative characterization of confluence for normally terminating programs.

Lemma 3

Let P be a CHR [annotated] normally terminating program. P is confluent if and only if for each pair of normal derivations $\sigma \mapsto^* \sigma_1^f \not\mapsto^*$ and $\sigma \mapsto^* \sigma_2^f \not\mapsto^*$, we have that $\sigma_1^f \equiv_{Fv(\sigma)} \sigma_2^f$.

Proof

(Only if) The proof is straightforward by definition of confluence.

(If) The proof is by contradiction. Assume that P is not confluent. Then, there exists a state σ such that $\sigma \mapsto^* \sigma_1$ and $\sigma \mapsto^* \sigma_2$ and for each pair of states σ'_f and σ''_f such that $\sigma_1 \mapsto^* \sigma'_f$ and $\sigma_2 \mapsto^* \sigma''_f$, we have that $\sigma'_f \not\equiv_{Fv(\sigma)} \sigma''_f$. In particular, since P is normally terminating, we have that there exists σ'_f and σ''_f such that $\sigma_1 \mapsto^* \sigma'_f \not\mapsto^*$, $\sigma_2 \mapsto^* \sigma''_f \not\mapsto^*$ and $\sigma'_f \not\equiv_{Fv(\sigma)} \sigma''_f$. Then, it is easy to check that there exist two normal derivation $\sigma \mapsto^* \sigma'_1 \not\mapsto^*$ and $\sigma \mapsto^* \sigma'_2 \not\mapsto^*$ such that $\sigma'_f \simeq \sigma'_1$ and $\sigma''_f \simeq \sigma'_2$. Since $\sigma'_f \not\equiv_{Fv(\sigma)} \sigma''_f$, by definition of \simeq , we have that $\sigma'_1 \not\equiv_{Fv(\sigma)} \sigma'_2$ and then we have a contradiction.

□

Then, we have the desired result.

Proposition 4

Let P be an annotated CHR program and let $cl \in P$ such that cl can be weakly safely replaced (by its unfolded version) in P . Moreover let

$$P' = (P \setminus \{cl\}) \cup Unf_P(cl).$$

If P is normally terminating then P' is normally terminating. Moreover, if P is normally terminating and confluent then P' is confluent too.

Proof

First, we prove that if P is normally terminating then P'' is normally terminating too, where

$$P'' = P \cup Unf_P(cl).$$

Then, we prove that if P'' is normally terminating then P' is normally terminating. Analogously if P is normally terminating and confluent and then the thesis.

- Assume that P is normally terminating. The proof of the normal termination of P'' follows by Proposition 6.
- Now, assume that P is normally terminating and confluent and by the contrary that P'' does not satisfy confluence.

By Lemma 3 and since by the previous result P'' is normally terminating, there exist a state σ and two normal derivations

$$\sigma \longrightarrow_{\omega'_i}^* \sigma'_f \not\rightarrow_{\omega'_i} \quad \text{and} \quad \sigma \longrightarrow_{\omega'_i}^* \sigma''_f \not\rightarrow_{\omega'_i}$$

in P'' such that $\sigma'_f \not\equiv_{Fv(\sigma)} \sigma''_f$.

Then, by using arguments similar to that given in Proposition 6 and since $P \subseteq P''$, we have that there exist two normal derivations

$$\sigma \longrightarrow_{\omega'_i}^* \sigma_1^f \not\rightarrow_{\omega'_i} \quad \text{and} \quad \sigma \longrightarrow_{\omega'_i}^* \sigma_2^f \not\rightarrow_{\omega'_i}$$

in P , where $\sigma'_f \simeq \sigma_1^f$ and $\sigma''_f \simeq \sigma_2^f$. Since by hypothesis P is confluent, we have that $\sigma_1^f \equiv_{Fv(\sigma)} \sigma_2^f$. Therefore, by Lemma 2 we have a contradiction to the assumption that there exist two states σ'_f and σ''_f as previously defined.

Now, we prove that if P'' is normally terminating then P' is normally terminating. Moreover we prove that if P'' is normally terminating and confluent then P' is confluent too and then the thesis.

- If P'' is normally terminating then, since $P' \subseteq P''$, we have that P' is normally terminating too.
- Now, assume that P'' is normally terminating and confluent and by the contrary that P' does not satisfy confluence. Moreover, assume that cl is of the form $r@H_1 \setminus H_2 \Leftrightarrow D \mid A; T$. By Lemma 3 and since by the previous result P' is normally terminating, there exist a state σ and two normal derivations

$$\sigma \longrightarrow_{\omega'_t}^* \sigma_1^f \not\rightarrow_{\omega'_t} \quad \text{and} \quad \sigma \longrightarrow_{\omega'_t}^* \sigma_2^f \not\rightarrow_{\omega'_t}$$

in P' such that $\sigma_1^f \not\equiv_{Fv(\sigma)} \sigma_2^f$.

Since $P' \subseteq P''$, we have that there exist two normal derivations

$$\sigma \longrightarrow_{\omega'_t}^* \sigma_1^f \quad \text{and} \quad \sigma' \longrightarrow_{\omega'_t}^* \sigma_2^f$$

in P'' . Then, since P'' is confluent and $P'' = P' \cup \{cl\}$ there exists $i \in [1, 2]$ such that $\sigma_i^f \longrightarrow_{\omega'_t} \sigma'$ in P'' by using the rule $cl \in (P'' \setminus P')$. In this case, by definition of weak safe replacement, there exists an unfolded rule $cl' \in Unf_P(cl)$ such that cl' is of the form

$$r@H_1 \setminus H_2 \Leftrightarrow D' \mid A'; T'$$

with $\mathcal{CT} \models D \Leftrightarrow D'$ and by construction $cl' \in P'$. Therefore $\sigma_i^f \longrightarrow_{\omega'_t} \sigma''$ in P' , by using the rule cl' , and then we have a contradiction.

□