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 used by most program transformation systems and consists of the replacement of a procedure call by its definition. While there is a large body of literature on the transformation and unfolding of sequential programs, very few papers have addressed this issue for concurrent languages. This paper defines an unfolding system for Constraint Handling Rules programs. We define an unfolding rule, show its correctness and discuss some conditions that 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.

KEYWORDS: CHR (Constraint Handling Rules), program transformation, unfolding, confluence, termination

1 Introduction

Constraint Handling Rules (CHR) (Frühwirth 1998, 2009; Frühwirth and Abdennadher 2003) is a concurrent, committed-choice language which was initially designed for writing constraint solvers, and is nowadays a general-purpose language. A CHR program is a (finite) set of guarded rules which allow to transform multi-sets 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 Holzbaur (2003), Frühwirth (2005), Sneyers et al. (2005), Tacchella et al. (2007), Tacchella (2008), and 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 the 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 (1984) more than 20 years ago, when considering logic concurrent languages it becomes quite difficult to define reasonable conditions that 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 that allows one to replace in a program a rule by its unfolded version while preserving qualified answers. This condition also preserves 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 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 that 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 $\omega_t$, 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 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 resulting programs are called annotated and we define their semantics by providing a (slightly) modified version of the semantics $\omega_t$, 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 $\omega_t$ (Duck et al. 2004) and the modified semantics $\omega'_t$ are given in Section 3. Section 4 defines the unfolding rule (without replacement) and proves its correctness. Section 5 discusses 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 that 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 to improve the readability of the paper.

A preliminary version of this paper appeared in Tacchella et al. (2007), and 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 \( p(t_1, \ldots, t_n) \), where \( p \) is some predicate symbol of arity \( n \geq 0 \) and \( (t_1, \ldots, t_n) \) is an \( n \)-tuple of terms. A term is (inductively) defined as a variable \( X \), or as \( f(t_1, \ldots, t_n) \), where \( f \) is a function symbol of arity \( n \geq 0 \) and \( t_1, \ldots, t_n \) are terms. \( \mathcal{F} \) 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 \( A \) with respect to 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. \( \text{Fv}(A) \) denotes the free variables appearing in \( A \).

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

A substitution is a mapping \( \theta : V \rightarrow \mathcal{F} \) such that \( \text{dom}(\theta) = \{ X \mid \theta(X) \neq X \} \) (domain of \( \theta \)) is finite; \( \varepsilon \) is the empty substitution: \( \text{dom}(\varepsilon) = \emptyset \).

The composition \( \theta \sigma \) of the substitutions \( \theta \) and \( \sigma \) is defined as the functional composition. A substitution \( \theta \) is idempotent if \( \theta \theta = \theta \). A renaming is a (non-idempotent) substitution \( \rho \) for which there exists the inverse \( \rho^{-1} \) 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 \( \mathcal{C} \mathcal{F} \) which describes their meaning. We also assume that the built-in constraints contain the predicate = which is described, as usual, by the Clark Equality Theory (Lloyd 1984) and the values \text{true} and \text{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
generical constraints). The capital versions will be used to denote multi-sets (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 that can be divided into the following three types: simplification, propagation, and simpagation rules. The first type of rules is used to rewrite CHR constraints into simpler ones, while the second type allows to add new redundant constraints that 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 \( H_1 \leftrightarrow H_2 \) 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 \( r @ i_1, \ldots, i_l \), where \( r \) is the name of a rule and \( i_1, \ldots, i_l \) 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:

\[ r \oplus H_1 \setminus H_2 \Leftrightarrow C | B ; T \]

where \( r \) is an identifier, \( H_1 \) and \( H_2 \) are sequences of user-defined constraints with \( H_1 \not\subseteq H_2 \) 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: \( chr(h\#i) =_{\text{def}} h \) and the overloaded function \( id(h\#i) =_{\text{def}} i \), (and \( id(r@i_1,\ldots,i_l) =_{\text{def}} \{i_1,\ldots,i_l\} \)), 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 that 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) 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:

\[
\text{Ann}(P) = \{ r \oplus H_1 \setminus H_2 \Leftrightarrow C | I(B); \emptyset \text{ such that } \}
\]

\[
r \oplus H_1 \setminus H_2 \Leftrightarrow C | B \in P
\}

Notation

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

\[ r \oplus H_2 \Leftrightarrow C | B(; T), \]

we write it as

\[ r \oplus H_2 \Leftrightarrow C | B(; T), \]
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 previously introduced notation for simplification and propagation rules. Moreover, if $C = \text{true}$, then $\text{true} \mid$ is omitted and if in an annotated rule the token store is empty, then 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 $cl$, $cl'$, … to denote (possibly annotated) rules and $cl_1$, $cl'_1$, … 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:

\[
\begin{align*}
  r_1 & @\text{root}(V), \text{same}(X, Y) \Rightarrow X = Y, X = V \mid \text{success}(V) \\
  r_2 & @\text{root}(V), \text{same}(X, Y) \iff X \neq Y \mid \text{root}(V), \text{same}(V, X), \text{path}(V, Y) \\
  r_3 & @\text{path}(I, J) \Rightarrow I = J \mid \text{true} \\
  r_4 & @\text{edge}(U, Z) \setminus \text{path}(I, J) \iff J = Z \mid \text{path}(I, U) \\
  r_5 & @\text{root}(V) \setminus \text{path}(I, J) \iff V = J, V \neq I \mid \text{false}
\end{align*}
\]

Then its annotated version $\text{Ann}(P)$ is defined as follows:

\[
\begin{align*}
  r_1 & @\text{root}(V), \text{same}(X, Y) \Rightarrow X = Y, X = V \mid \text{success}(V) #1; \emptyset \\
  r_2 & @\text{root}(V), \text{same}(X, Y) \iff X \neq Y \mid \text{root}(V) #1, \text{same}(V, X) #2, \text{path}(V, Y) #3; \emptyset \\
  r_3 & @\text{path}(I, J) \Rightarrow I = J \mid \text{true}; \emptyset \\
  r_4 & @\text{edge}(U, Z) \setminus \text{path}(I, J) \iff J = Z \mid \text{path}(I, U) #1; \emptyset \\
  r_5 & @\text{root}(V) \setminus \text{path}(I, J) \iff V = J, V \neq I \mid \text{false}; \emptyset
\end{align*}
\]

### 3 CHR operational semantics

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

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

In the following, given a (possibly annotated) rule $cl_1 = r@H_1 \setminus H_2 \iff C \mid B(; T)$, we denote by $\exists_{cl_1}$ the existential quantification $\exists_{Fr(H_1, H_2, C, B)}$. By an abuse of notation, when it is clear from the context, we will write $\exists_r$ instead of $\exists_{cl_r}$. 
Table 1. The transition system $T_{\omega_t}$ for the $\omega_t$ semantics.

| Rule       | Transition                                                                 |
|------------|----------------------------------------------------------------------------|
| Solve      | $c$ is a built-in constraint $\langle \{c\} \uplus G, S, C, T \rangle \xrightarrow{\omega_t} \langle G, S, C \land c, T \rangle$ |
| Introduce  | $h$ is a user-defined constraint $\langle \{h\} \uplus G, S, C, T \rangle \xrightarrow{\omega_t} \langle G, \{h\#n\} \uplus S, C, T \rangle$ |
| Apply      | $r \odot H_1 \setminus H_2 \Leftrightarrow D \mid B \in P$ $C \models D \Rightarrow \exists ((\text{chr}(H_1, H_2) = (H'_1, H'_2)) \land D)$ $\langle G, H_1 \uplus H_2 \uplus S, C, T \rangle \xrightarrow{\omega_t} \langle B \uplus G, H_1 \uplus S, (\text{chr}(H_1, H_2) = (H'_1, H'_2)) \land D \land C, T' \rangle$ |

where $r \odot \text{id}(H_1, H_2) \notin T$ and $T' = T \cup \{r \odot \text{id}(H_1, H_2)\}$

3.1 The semantics $\omega_t$

We describe the operational semantics $\omega_t$, introduced in Duck et al. (2004), by using a transition system $T_{\omega_t} = (Conf_t, \rightarrow_{\omega_t})$.

Configurations in $Conf_t$ are tuples of the form $\langle G, S, C, T \rangle$, where $G$, the goal store, is a multi-set 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 $\sigma, \sigma', \sigma_i, \ldots$ to represent configurations in $Conf_t$.

The goal store ($G$) contains all constraints to be executed. The CHR constraint store ($S$) is the set$^1$ of identified CHR constraints that can be matched with the head of the rules in 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 that can be used to number a CHR constraint.

Given a goal $G$, the initial configuration has the form $\langle G, \emptyset, \text{true}, \emptyset \rangle$.

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

The relation $\rightarrow_{\omega_t}$ (of the transition system $T_{\omega_t}$) 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)

$^1$ Note that sometimes we treat $S$ as a multi-set. This is the case, for example, of the transition rules, where considering $S$ as a multi-set simplifies the notation.
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 that 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 clashes of variable names.

It is clear from the rules 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 \( \omega'_t \)

We now define the semantics \( \omega'_t \) that considers annotated rules. This semantics differs from \( \omega_t \) in two aspects.

First, in \( \omega'_t \) 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 used to keep track of the propagation rules that are used to unfold the body of \( cl \). Also note that here, different from the case of the propagation history in \( \omega_t \), 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 \( \omega'_t \) formally, we need a function \( \text{inst} \) which updates the formal identifiers of a rule to the actual computation ones. Such a function is defined as follows.

**Definition 4**

Let \( \text{Token} \) be the set of all possible token sets and let \( \mathbb{N} \) be the set of natural numbers. We denote by \( \text{inst} : \text{Goals} \times \text{Token} \times \mathbb{N} \rightarrow \text{Goals} \times \text{Token} \times \mathbb{N} \) the function such that \( \text{inst}(B, T, n) = (B', T', m) \), 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 now describe the operational semantics \(\omega'\) for annotated CHR programs by using, as usual, a transition system

\[ T_{\omega'} = (Conf'_{\omega'}, \rightarrow_{\omega'}). \]

Configurations in \(Conf'_{\omega'}\) are tuples of the form \(\langle S, C, T \rangle_n\) with the following meaning. \(S\) is the set\(^2\) 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 that was used to number the CHR constraints in \(S\).

Given a goal \(G\), the initial configuration has the form

\[ \langle I(G), \text{true}, \emptyset \rangle_m, \]

where \(m\) is the number of CHR constraints in \(G\) and \(I\) is the function that associates the identifiers with the CHR constraints in \(G\). A failed configuration has the form \(\langle S, \text{false}, T \rangle_n\).

A final configuration has either failed or it has the form \(\langle S, C, T \rangle_n\) (with \(C \neq false\)) when it represents a successful termination, since there are no more applicable rules.

The relation \(\rightarrow_{\omega'}\) (of the transition system \(T_{\omega'}\)) 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 \(r \circledast H_1 \setminus H_2 \leftrightarrow D \mid B; T_r\) 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 (also taking into account the matching mentioned before); \(r \circledast \text{id}(H_1, H_2) \notin T\). These conditions are equal to those already seen for **Apply**. Moreover, analogous 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 \(r \circledast \text{id}(H_1, H_2)\)) where each identifier is suitably incremented by the \(\text{inst}\) 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 name clashes.

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

\(^2\) Also in this case, sometimes we treat \(S\) as a multi-set. See the previous footnote.
Example 2

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

\[
\begin{align*}
&\text{root}(V), \text{same}(X, Y) \Rightarrow X = Y, X = V \mid \text{success}(V)\#1; 0 \\
&\text{root}(V), \text{same}(X, Y) \Leftrightarrow X \neq Y \mid \text{root}(V)\#1, \text{same}(V, X)\#2, \text{path}(V, Y)\#3; 0 \\
&\text{root}(V), \text{same}(X, Y) \Leftrightarrow X \neq Y, V = X \mid \text{root}(V)\#1, \text{same}(V, X)\#2, \text{path}(V, Y)\#3, \\
&\hspace{2cm} \text{success}(I)\#4, V = I, V = J, X = L; \{r_1 @ 1, 2\} \\
&\text{root}(V), \text{same}(X, Y) \Leftrightarrow X \neq Y, V \neq X \mid \text{path}(V, Y)\#3, \text{root}(I)\#4, \text{same}(J, L)\#5, \\
&\hspace{2cm} I = J, V = L, X = 0; \{r_2 @ 1, 2\} \\
&\text{root}(V), \text{same}(X, Y) \Leftrightarrow X \neq Y, V = Y \mid \text{root}(V)\#1, \text{same}(V, X)\#2, \\
&\hspace{2cm} \text{path}(V, Y)\#3, V = I, Y = J; \{r_3 @ 3\} \\
&\text{path}(I, J) \Rightarrow I = J \mid \text{true}; 0 \\
&\text{edge}(U, Z) \setminus \text{path}(I, J) \Leftrightarrow J = Z \mid \text{path}(I, U)\#1; 0 \\
&\text{edge}(U, Z) \setminus \text{path}(I, J) \Leftrightarrow J = Z, I = U \mid \text{path}(I, U)\#1, I = X, U = Y; \{r_3 @ 1\} \\
&\text{root}(V) \setminus \text{path}(I, J) \Rightarrow J = V, V \neq I \mid \text{false}; 0
\end{align*}
\]

we obtain the following derivation

\[
\begin{align*}
&\langle \text{root}(a)\#1, \text{same}(b, c)\#2, \text{edge}(a, b)\#3, \text{edge}(a, c)\#4, \text{edge}(d, c)\#5), \text{true}, 0 \rangle_s \rightarrow^*_{\omega'} \\
&\langle \text{path}(V_1, Y_1)\#6, \text{root}(I_1)\#7, \text{same}(J_1, L_1)\#8, I_1 = V_1, J_1 = V_1, L_1 = X_1, \\
&\hspace{2cm} \text{edge}(a, b)\#3, \text{edge}(a, d)\#4, \text{edge}(d, c)\#5), \\
&\hspace{2cm} (a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \{r_2 @ 1, 2\}\rangle_s \rightarrow^*_{\omega'} \\
&\langle \text{path}(V_1, Y_1)\#6, \text{root}(I_1)\#7, \text{same}(J_1, L_1)\#8, \text{edge}(a, b)\#3, \text{edge}(a, c)\#4, \text{edge}(d, c)\#5), \\
&\hspace{2cm} (I_1 = V_1, J_1 = V_1, L_1 = X_1, a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \{r_2 @ 1, 2\}\rangle_s \rightarrow^*_{\omega'} \\
&\langle \text{root}(V_2)\#9, \text{same}(V_2, X_2)\#10, \text{path}(V_2, Y_2)\#11, \text{success}(I_2)\#12, \\
&\hspace{2cm} V_2 = I_2, V_2 = J_2, X_2 = L_2, \text{path}(V_1, Y_1)\#6, \text{edge}(a, b)\#3, \text{edge}(a, c)\#4, \text{edge}(d, c)\#5), \\
&\hspace{2cm} (V_2 = I_1, X_2 = J_1, Y_2 = L_1, X_2 \neq Y_2, V_2 = X_2, I_1 = V_1, J_1 = V_1, L_1 = X_1, \\
&\hspace{4cm} a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \{r_2 @ 1, 2, r_2 @ 7, 8, r_1 @ 9, 10\}\rangle_{12} \rightarrow^*_{\omega'} \\
&\langle \text{root}(V_2)\#9, \text{same}(V_2, X_2)\#10, \text{path}(V_2, Y_2)\#11, \text{success}(I_2)\#12, \\
&\hspace{2cm} \text{path}(V_1, Y_1)\#6, \text{edge}(a, b)\#3, \text{edge}(a, c)\#4, \text{edge}(d, c)\#5), \\
&\hspace{2cm} (V_2 = I_2, V_2 = J_2, X_2 = L_2, V_2 = I_1, X_2 = J_1, Y_2 = L_1, X_2 \neq Y_2, V_2 = X_2, I_1 = V_1, J_1 = V_1, \\
&\hspace{4cm} L_1 = X_1, a = V_1, b = X_1, c = Y_1, X_1 \neq Y_1, V_1 \neq X_1), \{r_2 @ 1, 2, r_2 @ 7, 8, r_1 @ 9, 10\}\rangle_{12} \rightarrow^*_{\omega'}
\end{align*}
\]
Let \( P \) be a CHR program and let \( G \) be a goal. The set \( \mathcal{Q}_P(G) \) of qualified answers for the query \( G \) in the program \( P \) is defined as follows:

\[
\mathcal{Q}_P(G) = \{ \exists_{F \in \mathcal{G}}(\text{chr}(K) \land D) \mid \mathcal{C}^T \not\models D \leftrightarrow \text{false} \text{ and } \langle G, \emptyset, \text{true}, \emptyset \rangle^* \rightarrow_{\omega} \langle \emptyset, K, D, T \rangle_n \not\models_{\omega} \}
\]

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

**Definition 5 (Qualified Answers)**

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

\[
\mathcal{Q}_P^\prime(G) = \{ \exists_{F \in \mathcal{G}}(\text{chr}(K) \land D) \mid \mathcal{C}^T \not\models D \leftrightarrow \text{false} \text{ and } \langle I(G), \text{true}, \emptyset \rangle_m \rightarrow^*_{\omega} \langle K, D, T \rangle_n \not\models_{\omega} \}
\]

From the previous transition systems we can obtain a notion of observable property of CHR computations that will be used 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 that have not been “solved” (i.e., transformed by rule applications into built-in constraints). Formally qualified answers are defined as follows.

**Definition 6 (Qualified Answers for Annotated Programs)**

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

\[
\mathcal{Q}_P^\prime(G) = \{ \exists_{F \in \mathcal{G}}(\text{chr}(K) \land D) \mid \mathcal{C}^T \not\models D \leftrightarrow \text{false} \text{ and } \langle I(G), \text{true}, \emptyset \rangle_m \rightarrow^*_{\omega} \langle K, D, T \rangle_n \not\models_{\omega} \}
\]
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 $G$,

$$2\mathcal{A}_P(G) = 2\mathcal{A}_{Ann(P)}(G)$$

holds.

4 The unfolding rule

In this section, we define the unfold operation for CHR simpagation rules. As a particular case, we also obtain 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 the body of the rule $cl_r$ imply that the head $H$ of $cl_v$, instantiated by a substitution $\theta$, 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 $\theta$ and without those constraints that are entailed by the built-in constraints that 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 respectively on the fact that $cl_v$ is a simplification, a simpagation, or a propagation rule; (4) as for the local token store $Tr$ 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:

$$r_1 @ b(Acc1, Bal1), b(Acc2, Bal2), t(Acc1, Acc2, Amount) \Leftarrow Acc1 \neq Acc2 \mid b(Acc1, Bal1), b(Acc2, Bal2), w(Acc1, Amount), d(Acc2, Am)$$

$$r_2 @ b(Acc, Bal), d(Acc, Am) \Leftarrow b(Acc, B), B = Bal + Am$$

$$r_3 @ b(Acc', Bal'), w(Acc', Am') \Leftarrow Bal' > Amount' \mid b(Acc', B'), B' = Bal' - Am'$$
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 obtain the new clause \( cl'_{r_1} : \)

\[
\begin{align*}
  r_1 @ b(Acc1, Bal1, B, Acc2, Bal2, Amount) \iff Acc1 \neq Acc2 | \\
  b(Acc1, Bal1, w(Acc1, Amount), b(Acc, B), B = Bal + Am, Acc2 = Acc, Bal2 = Bal, Amount = Am.
\end{align*}
\]

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

\[
\begin{align*}
  r_3 @ b(Acc1, Bal1, B, Acc2, Bal2, Amount) \iff \\
  Acc1 \neq Acc2, Bal1 > Amount | b(Acc, B), B = Bal + Am, Acc2 = Acc, Bal2 = Bal, Amount = Am, b(Acc', B'), B'' = Bal' - Am', Acc1 = Acc', Bal1 = Bal', Amount = Am'.
\end{align*}
\]

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

**Definition 7**

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

\[
clean : Goals \times Token \to Token
\]

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,

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

**Definition 8 (UNFOLD)**

Let \( P \) be an annotated CHR program and let \( cl_r, cl_i \in P \) be the two following annotated rules

\[
\begin{align*}
  r@H_1 \backslash H_2 \iff D | K, S_1, S_2, C; T \text{ and } \\
  v@H_1' \backslash H_2' \iff D' | B; T'
\end{align*}
\]

respectively, where \( C \) is the conjunction of all the built-in constraints in the body of \( cl_r \). Let \( \theta \) be a substitution such that \( \text{dom}(\theta) \subseteq Fv(H_1', H_2') \) and \( \mathcal{G} \models (C \land D) \to \text{chr}(S_1, S_2) = (H_1', H_2')\theta \). Furthermore, let \( m \) be the greatest identifier that appears in the rule \( cl_r \) and let \( (B_1, T_1, m_1) = \text{inst}(B, T', m) \). Then the _unfolded_ rule is:

\[
\begin{align*}
  r@H_1 \backslash H_2 \iff D, (D''|\theta) | K, S_1, B_1, C, \text{chr}(S_1, S_2) = (H_1', H_2'); T''
\end{align*}
\]

where \( v@\text{id}(S_1, S_2) \notin T, V = \{ d \in D' \mid \mathcal{G} \models C \land D \to d\theta \}, D'' = D' \setminus V, Fv(D''|\theta) \subseteq Fv(H_1, H_2), \) the constraint \((D,(D''|\theta))\) is satisfiable and

- if \( H_2' = e \) then \( T'' = T \cup T_1 \cup \{ v@\text{id}(S_1) \} \)
- if \( H_2' \neq e \) 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 \( \mathcal{G} \models C \land D \to d\theta \) for each \( d \in V \). Moreover, as shown in the following, all the results in the paper are independent of the choice of the substitution \( \theta \) which satisfies the conditions of Definition 8. Finally, we use the function \( \text{inst} \) (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. Also note that the condition on the token store is needed to obtain a correct rule. Consider for example a ground annotated program

\[
P = \{ \begin{align*}
    r_1 @ h &\iff k#1 \\
r_2 @ k &\Rightarrow s#1 \\
r_3 @ s, s &\iff q#1
\end{align*} \}
\]

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. 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 \(r_1 @ h \iff k#1, s#2\) unfold using \(cl_{r_2}\) \(\Rightarrow r_1 @ h \iff k#1, s#2, s#3\) unfold using \(cl_{r_3}\) \(\Rightarrow r_1 @ h \iff k#1, q#4.\) So, starting from the constraint \(h\) we could obtain the qualified answer \((k, q),\) which is not possible in the original program (the rule obtained after the wrongly applied unfolding rule is underlined).

As mentioned previously, 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

\[
\begin{align*}
    r_1 @ f(X, Y), f(Y, Z), f(Z, W) &\iff g(X, Z)#1, f(Z, W)#2, gs(Z, X)#3 \\
    r_2 @ g(U, V), f(V, T) &\iff gg(U, T)#1 \\
    r_3 @ g(U, V), f(V, T) &\Rightarrow gg(U, T)#1 \\
    r_4 @ g(J, L), f(L, N) &\iff gg(J, N)#1
\end{align*}
\]

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 \(cl_{r_4}\).

Now we unfold the body of rule \(cl_{r_1}\) by using the simplification rule \(cl_{r_2}\). We use the \(inst\) function \(inst(gg(U, T)#1, 0, 3) = (gg(U, T)#4, 0, 4).\) So the new unfolded rule is as follows:

\[
\begin{align*}
    r_1 @ f(X, Y), f(Y, Z), f(Z, W) &\iff gs(Z, X)#3, gg(U, T)#4, X = U, Z = V, W = T.
\end{align*}
\]

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 \(inst(gg(U, T)#1, 0, 3) = (gg(U, T)#4, 0, 4)\) and then the new unfolded rule is

\[
\begin{align*}
    r_1 @ f(X, Y), f(Y, Z), f(Z, W) &\iff g(X, Z)#1, f(Z, W)#2, gs(Z, X)#3, \\
    &\iff gg(U, T)#4, X = U, Z = V, W = T; \{r_3 @ 1, 2\}.
\end{align*}
\]

---

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.
Finally, we unfold the body of rule \( clr_1 \) by using the simpagation rule \( clr_4 \). As before, the function

\[
\text{inst}(gg(J, N)#1, 0, 3) = (gg(J, N)#4, 0, 4)
\]

is computed. The new unfolded rule is:

\[
\begin{align*}
& r_1 \land f(X, Y), f(Y, Z), f(Z, W) \iff g(X, Z)#1, gs(Z, X)#3, \\
& \quad gg(J, N)#4, X = J, Z = L, W = J.
\end{align*}
\]

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

**Example 5**

Consider the program consisting of the following rules

\[
\begin{align*}
& r_1 \land f(X, Y), f(Y, Z), f(Z, W) \iff X = Adam, Y = Seth | \\
& \quad g(X, Z)#1, f(Z, W)#2, gs(Z, X)#3, Z = Enosh \\
& r_2 \land g(U, V), f(V, T) \Rightarrow U = Adam, V = Enosh | gg(U, T)#1, T = Kenan \\
& r_3 \land g(J, L) \land f(L, N) \iff J = Adam, L = Enosh | gg(J, N)#1, N = Kenan
\end{align*}
\]

that, as usual, we call \( clr_1, clr_2, \) and \( clr_3 \), 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; and 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 \( clr_1 \) by using \( clr_3 \) we have

\[
\begin{align*}
& r_1 \land f(X, Y), f(Y, Z), f(Z, W) \iff X = Adam, Y = Seth | \\
& \quad g(X, Z)#1, gs(Z, X)#3, Z = Enosh, \\
& \quad gg(J, N)#4, N = Kenan, X = J, Z = L, W = N.
\end{align*}
\]

Moreover, when \( clr_2 \) is considered to unfold \( clr_1 \), we obtain

\[
\begin{align*}
& r_1 \land f(X, Y), f(Y, Z), f(Z, W) \iff X = Adam, Y = Seth | \\
& \quad g(X, Z)#1, f(Z, W)#2, gs(Z, X)#3, Z = Enosh, \\
& \quad gg(U, T)#4, T = Kenan, X = U, Z = V, W = T; \{r_2 \land 1, 2\}.
\end{align*}
\]

Note that \( U = Adam, V = Enosh \), which is the guard of the rule \( clr_2 \), is not added to the guard of the unfolded rule because \( U = Adam \) is entailed by the guard of \( clr_1 \) and \( V = Enosh \) is entailed by the built-in constraints in the body of \( clr_1 \) by considering also the binding provided by the parameter passing (analogously for \( clr_3 \)).

**Example 6**

The program \( P' \) of 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 noting 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 given 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 $G$, $QA'_P(G) = QA'_P'(G)$ holds.

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

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

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 $G$, $QA_P(G) = QA'_P'(G)$.

Proof
The proof follows from Propositions 1 and 2. □

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 the 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 to allow such a replacement, as we do later. The second source of difficulties consists in the pattern matching mechanism 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, whereas 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, whereas if we move the pattern matching to the guard we could still unfold the rule (under suitable conditions).

Finally, we have the problem of 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 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 that were not in the original program. This is a peculiarity of CHR and it is different from what happens in Prolog.

The next section 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

\[
P = \{ r_1@p(Y) ⇔ q(Y), s(Y) \\
r_2@q(Z) ⇔ Z = a|\text{true} \\
r_3@s(V) ⇔ V = a \}
\]

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 \( r_1@p(Y) ⇔ q(Y), s(Y) \) by using the rule \( r_2@q(Z) ⇔ Z = a|\text{true} \) returns the new rule \( r_1@p(Y) ⇔ Y = a | s(Y), Y = Z \). Now the program

\[
P' = \{ r_1@p(Y) ⇔ Y = a | s(Y), Y = Z \\
r_2@q(Z) ⇔ Z = a|\text{true} \\
r_3@s(V) ⇔ V = a \}
\]

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

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
Unfolding for CHR programs

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

\[ P = \{ r_1 \circ p(X,Y) \iff q(Y,X) \]
\[ r_2 \circ q(W,a) \iff W = b \]
\[ r_3 \circ q(J,T) \iff J = d \} \]

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 \( r_1 \circ p(X,Y) \iff Y = J, X = T, J = d \) in \( P \), substitutes the original one (note that other unfoldings are not possible, in particular the rule \( r_2 \circ q(W,a) \iff W = b \) can not be used to unfold \( r_1 \circ p(X,Y) \iff q(Y,X) \)).

\[ P' = \{ r_1 \circ p(X,Y) \iff Y = J, X = T, J = d \]
\[ r_2 \circ q(W,a) \iff W = b \]
\[ r_3 \circ q(J,T) \iff J = d \} \]

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

\[ P = \{ r \circ p(Y) \iff q(Y), h(b) \]
\[ v \circ q(Z), h(V) \iff Z = V \} \]

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

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

\[ r \circ p(Y) \iff Y = Z, V = b, Z = V. \]

Now the program

\[ P' = \{ r \circ p(Y) \iff Y = Z, V = b, Z = V \]
\[ v \circ q(Z), h(V) \iff Z = V \} \]

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 \( (X = a) \in \mathcal{A}'_P(G) \), while \( (X = a) \notin \mathcal{A}'_{P'}(G) \).
5.2 A condition for safe rule replacement

We have identified some conditions that 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 $cl_v$ that 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 set contains pairs: for each pair the first component is a rule that can be used to unfold $cl_r$, while the second component 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 also contains 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

$$
\begin{align*}
  r \circ @ H_1 \setminus H_2 & \iff D \mid A; T \\
  v \circ @ H'_1 \setminus H'_2 & \iff D' \mid B; T'
\end{align*}
$$

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. $\forall (cl_v, (i_1, \ldots, i_n)) \in U^+_P(cl_r)$ 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 $(i_1, \ldots, i_n)$.

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 $D \land C$ is satisfiable, $\mathcal{C} \mathcal{T} \models (D \land C) \rightarrow \exists cl_r ((\operatorname{chr}(A_1, A_2) = (H'_1, H'_2)) \land D')$, $v \circ @ \operatorname{id}(A_1, A_2) \notin T$, and $(cl_v, \operatorname{id}(A_1, A_2)) \notin U^+_P(cl_r)$, or

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

Some explanations are in order here.
The set \( U^+ \) contains all the couples composed of the 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 now consider the set \( U^\# \). The conjunction of built-in constraints \( C \) represents a generic set of built-in constraints (such a set can naturally be equal to every possible built-in constraint store that can be generated by a real computation before the application of rule \( cl_r \)); the condition \( Fv(C) \cap Fv(cl_r) = \emptyset \) is required to avoid free variable capture, it represents the renaming (with fresh variables) of rule \( cl_r \) with respect to the computation before the use of \( cl_r \) 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 \( \forall \mathcal{F} \models (D \land C) \rightarrow \exists cl_r ((\text{chr}(A_1,A_2) = (H_1',H_2')) \land D') \) and \( \forall \mathcal{F} \models (D \land C) \rightarrow \exists cl_r ((\text{chr}(k) = h) \land D') \) secure that a strong enough built-in constraint is provided by the computation before the application of rule \( cl_r \); finally, the condition \( (cl_r, id(A_1,A_2)) \notin U^+_P(cl_r) \) 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 the one described in Example 8. The second kind, due to Condition 2b in Definition 9, indicates a multiple heads problem similar to the one given 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 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 \( cl \in P \),

\[
\text{Unf}_P(cl)
\]

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 \( r@H_1 \setminus H_2 \Leftarrow D \mid A; T \), such that the following holds

(i) \( U^\#_P(cl_r) = \emptyset \),

(ii) \( U^+_P(cl_r) \neq \emptyset \) and

(iii) for each \( r@H_1 \setminus H_2 \Leftarrow D' \mid A'; T' \in \text{Unf}_P(cl_r) \) we have that \( \forall \mathcal{F} \models D \leftrightarrow D' \).

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 \( cl_r \) only if \( cl_r \) can be unfolded with \( cl_r \) 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 \( C \mathcal{T} \models D \leftrightarrow D' \) 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 of the particular substitution \( \theta \) chosen in Definition 8 to define the unfolding of the rule

\[
\begin{align*}
r \circ H_1 \setminus H_2 & \Leftrightarrow D | K, S_1, S_2, C; T \\
v \circ H_1 \setminus H_2 & \Leftrightarrow D' | B; T'
\end{align*}
\]

In fact, let us assume that there exist two substitutions \( \theta \) and \( \gamma \) that satisfy the conditions of Definition 8. Then \( C \mathcal{T} \models (C \land D) \rightarrow (d\theta \leftrightarrow d\gamma) \) for each \( d \in D' \).

Therefore, if \( V = \{ d \in D' \mid C \mathcal{T} \models C \land D \rightarrow d\theta \} \) and \( W = \{ d \in D' \mid C \mathcal{T} \models C \land D \rightarrow d\gamma \} \), we have \( V = W \) 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 safe replacement.

**Example 10**

Consider the program \( P \) consisting of the following four rules

\[
\begin{align*}
r_1 \circ p(X, Y, Z) & \Leftrightarrow r(b, b, Z) \#_1, s(Z, b, a) \#_2, q(X, f(Z), a) \#_3, r(g(X, b), f(a), f(Z)) \#_4; 0 \\
r_2 \circ q(V, U, W), r(g(V, b), f(W), U) & \Leftrightarrow W = a | s(V, U, W) \#_1, r(U, U, V) \#_2; 0 \\
r_3 \circ r(M, M, N), s(N, M, a) & \Leftrightarrow p(M, N, N) \#_1; 0 \\
r_4 \circ s(L, J, I) & \Rightarrow I = L; 0
\end{align*}
\]

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

\[
\begin{align*}
U_P^+(cl_{r_1}) & = \{(cl_{r_2}, (3, 4)), (cl_{r_3}, (1, 2)), (cl_{r_4}, (2))\} \\
U_P^+(cl_{r_1}) & = \emptyset.
\end{align*}
\]

Moreover

\[
\begin{align*}
U_P^+(cl_{r_1}) &= \{ r_1 \circ p(X, Y, Z) \Leftrightarrow r(b, b, Z) \#_1, s(Z, b, a) \#_2, s(V, U, W) \#_5, r(U, U, V) \#_6, \\
& \quad X = V, U = f(Z), W = a; 0 \\
& \quad r_1 \circ p(X, Y, Z) \Leftrightarrow q(X, f(Z), a) \#_3, r(g(X, b), f(a), f(Z)) \#_4, p(M, N, N) \#_5, \\
& \quad M = b, N = Z; 0 \\
& \quad r_1 \circ p(X, Y, Z) \Leftrightarrow r(b, b, Z) \#_1, s(Z, b, a) \#_2, q(X, f(Z), a) \#_3, \\
& \quad r(g(X, b), f(a), f(Z)) \#_4, I = L, Z = L, b = J, a = I; \{r_4 \circ 2\} \}
\end{align*}
\]

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

\[
P_1 = (P \setminus \{cl_1\}) \cup U_P^+(cl_{r_1}),
\]
where $P_1$ is the program

\[
\begin{align*}
{r_1} @ p(X, Y, Z) & \iff r(b, b, Z) \# 1, s(Z, b, a) \# 2, s(V, U, W) \# 5, r(U, U, V) \# 6, \\
X = V, U = f(Z), W = a; \emptyset \\
{r_2} @ q(V, U, W), r(g(V, b), f(W), U) & \iff W = a | s(V, U, W) \# 1, r(U, U, V) \# 2; \emptyset \\
r_3 @ s(M, N, a) & \iff p(M, N, N) \# 1; \emptyset \\
r_4 @ s(L, J, I) & \Rightarrow I = L; \emptyset. 
\end{align*}
\]

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

**Theorem 1**

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

$$P' = (P \setminus \{cl\}) \cup \text{Unf}_P(cl).$$

Then $\mathcal{A}''_P(G) = \mathcal{A}''_{P'}(G)$ 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. A **U-sequence** of programs starting from $P$ is a sequence of annotated CHR programs $P_0, \ldots, P_n$, such that

- $P_0 = P$
- $P_{i+1} = (P_i \setminus \{cl^i\}) \cup \text{Unf}_{P_i}(cl^i)$,

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

**Example 11**

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

$$P_2 = (P_1 \setminus \{cl_2\}) \cup \text{Unf}_{P_1}(cl_2),$$

where $P_2$ is the program

\[
\begin{align*}
{r_1} @ p(X, Y, Z) & \iff r(b, b, Z) \# 1, s(Z, b, a) \# 2, s(V, U, W) \# 5, r(U, U, V) \# 6, \\
X = V, U = f(Z), W = a; \emptyset \\
{r_2} @ q(V, U, W), r(g(V, b), f(W), U) & \iff W = a | s(V, U, W) \# 1, r(U, U, V) \# 2; \emptyset \\
r_3 @ s(M, N, a) & \iff p(M, N, N) \# 1; \emptyset \\
r_4 @ s(L, J, I) & \Rightarrow I = L; \emptyset. 
\end{align*}
\]
Then from Theorem 1 and Proposition 1, we have the following.

**Corollary 2**
Let \( P \) be a program and let \( P_0, \ldots, P_n \) be an U-sequence starting from \( \text{Ann}(P) \). Then \( 2.A_P(G) = 2.A'_{P_n}(G) \) 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 \text{Solve} (\text{Solve}') transitions are applied as soon as possible, as specified by Definition 14. Moreover, we prove that our unfolding also preserves 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 that has no built-in constraints in the first component.

**Definition 13 (Built-in free configuration)**
Let \( \sigma = \langle G, S, D, T \rangle_o \in \text{Conf}_f \) (\( \sigma = \langle G, D, T \rangle_o \in \text{Conf}'_f \)). The configuration \( \sigma \) is built-in free if \( G \) is a multi-set 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 \( \delta \) be a derivation in \( P \). We say that \( \delta \) is normal if, for each configuration \( \sigma \) in \( \delta \), a transition \text{Apply} (\text{Apply}') is used on \( \sigma \) only if \( \sigma \) is built-in free.

Note that, by definition, given a CHR program \( P \), \( 2.A(P) \) 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 \emph{terminating} if there are no infinite derivations. A (possibly annotated) CHR program \( P \) is called \emph{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 given in the Appendix.

**Proposition 3 (Normal Termination)**
Let \( P \) be a CHR program and let \( P_0, \ldots, P_n \) be a 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 guard elimination in the unfolding. This is shown by the following example.
**Example 12**

Let us consider the following program:

\[ P = \{ \mathit{r_1} \mathbin{@} \mathit{p}(X) \iff X = a, q(X) \}
\]

\[ \quad \mathit{r_2} \mathbin{@} \mathit{q}(Y) \iff Y = a \mid r(Y) \]

\[ \quad \mathit{r_3} \mathbin{@} \mathit{r}(Z) \iff Z = d \mid p(Z) \} \]

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

\[ \mathit{r_2} \mathbin{@} \mathit{q}(Y) \iff Y = a \mid r(Y) \]

to unfold \( \mathit{r_1} \mathbin{@} \mathit{p}(X) \iff X = a, q(X) \) (with replacement) we obtain the following program \(P'\):

\[ P' = \{ \mathit{r_1} \mathbin{@} \mathit{p}(X) \iff X = a, X = Y, r(Y) \}
\]

\[ \quad \mathit{r_2} \mathbin{@} \mathit{q}(Y) \iff Y = a \mid r(Y) \]

\[ \quad \mathit{r_3} \mathbin{@} \mathit{r}(Z) \iff Z = d \mid p(Z) \} \]

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

\[ \langle (X = a, p(Z)\#3), (V = d, V = X, X = Y, Y = Z), \emptyset \rangle \]

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

The next property that 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 that introduces some specific notation for renaming of indexes.

**Definition 16**

Let \(j_1, \ldots, j_o\) be distinct identification values.

- A renaming of identifiers is a substitution of the form \([j_1/i_1, \ldots, j_o/i_o]\), where \(i_1, \ldots, i_o\) is a permutation of \(j_1, \ldots, j_o\).
- Given an expression \(E\) and a renaming of identifiers \(\rho = [j_1/i_1, \ldots, j_o/i_o]\), \(E\rho\) 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 \(\rho\) and \(\rho'\) 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 \(\rho, \rho', \ldots\) to denote renamings.

Now we need the following definition introducing a form of equivalence between configurations, which is a slight modification of the one in Raiser \textit{et al.} (2009), since it 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 $\text{Conf}_r$ and $\text{Conf}'_r$ with the same meaning.

**Definition 17**

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

- $\langle d \land G, S, C, T \rangle_n \equiv_V \langle G, S, d \land C, T \rangle_n$,
- $\langle G, S, X = t \land C, T \rangle_n \equiv_V \langle G[X/t], S[X/t], X = t \land C, T \rangle_n$,
- 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 $W = Fv(C) \setminus (Fv(G) \cup V)$, $U = Fv(C') \setminus (Fv(G) \cup V)$, and $CT \models \exists W C \leftrightarrow \exists U C'$ then $\langle G, S, C, T \rangle_n \equiv_V \langle G, S, C', T \rangle_n$,
- $\langle G, S, \text{false}, T \rangle_n \equiv_V \langle G', S', \text{false}, T' \rangle_m$,
- $\langle G, S, C, T \rangle_n \equiv_V \langle G, S, \rho, C, T, \rho \rangle_m$ for each renaming of identifiers $\rho$
- $\langle G, S, C, T \rangle_n \equiv_V \langle G, S, C, \text{clean}(S, T) \rangle_n$.

We can define the equivalence $\equiv_V$ between configurations in $\text{Conf}'_r$ in an analogous way.

**Definition 18**

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

- $\langle d \land G, C, T \rangle_n \equiv_V \langle G, d \land C, T \rangle_n$,
- $\langle G, X = t \land C, T \rangle_n \equiv_V \langle G[X/t], X = t \land C, T \rangle_n$,
- 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 $\langle G, C, T \rangle_n \equiv_V \langle G, C', T \rangle_n$,
- $\langle G, \text{false}, T \rangle_n \equiv_V \langle G', \text{false}, T' \rangle_m$,
- $\langle G, C, T \rangle_n \equiv_V \langle G, \rho, C, T, \rho \rangle_m$ for each renaming of identifiers $\rho$
- $\langle G, C, T \rangle_n \equiv_V \langle G, C, \text{clean}(G, T) \rangle_n$.

By definition of $\equiv_V$, it is straightforward to check that if $\sigma, \sigma' \in \text{Conf}_r(\text{Conf}'_r)$, $V$ 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 given in Raiser et al. (2009), since it considers also the cleaning of the token store.

In the following $\mapsto^*$ means either $\mapsto_{\omega}$ or $\mapsto_{\omega'}$. 
**Definition 19 (Confluence)**

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

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 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:

\[
P = \{ \begin{array}{l}
r_1 @ p \iff q \\
r_2 @ p \iff r \\
r_3 @ r \iff r, s \\
r_4 @ q \iff r, s \\
\end{array} \}
\]

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 \( r_3 \) itself (with safe rule replacement) we obtain the following program \( P' \):

\[
P' = \{ \begin{array}{l}
r_1 @ p \iff q \\
r_2 @ p \iff r \\
r_3 @ r \iff r, s, s \\
r_4 @ q \iff r, s \\
\end{array} \}
\]

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

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

and there exist no states \( \sigma_1 \) and \( \sigma'_1 \) such that \( \sigma \xrightarrow{*_{\omega'_c}} \sigma_1 \) and \( \sigma' \xrightarrow{*_{\omega'_c}} \sigma'_1 \) in \( P' \), where \( \sigma_1 \equiv_0 \sigma'_1 \).

Note that the program in the 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 \( (cl_r, (i_1, \ldots, i_n)) \in U_{+}^{+}(cl_r) \) we have \( cl_r \neq cl_r \).

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 \( \text{NRU-sequence} \) of programs starting from \( P \) is a sequence of annotated CHR programs \( P_0, \ldots, P_n \) such that

\[
P_0 = P \quad \text{and} \quad P_{i+1} = (P_i \setminus \{cl_i^l\}) \cup \text{Unf}_{P_i}(cl_i),
\]

where \( i \in [0, n - 1] \), \( cl_i^l \in P_i \) and can be non-recursively safely replaced in \( P_i \).

Theorem 2
Let \( P \) be a CHR program and let \( P_0, \ldots, P_n \) be an NRU-sequence starting from \( P_0 = \text{Ann}(P) \). \( P \) satisfies confluence if and only if \( P_n \) satisfies confluence too.

6 Weak safe rule replacement

In this section we consider only the programs that are normally terminating and confluent. For this class of programs we give a condition for rule replacement which is much weaker than the one used in the previous section and 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 the one in \( cl_r \).

Definition 22 (Weak safe rule replacement)
Let \( P \) be an annotated CHR program, and let \( r \@ H_1 \setminus H_2 \iff D \mid A; T \in P \) be a rule such that there exists

\[
r \@ H_1 \setminus H_2 \iff D' \mid A'; T' \in \text{Unf}_P(r \@ H_1 \setminus H_2 \iff D \mid A; T)
\]

with \( \mathcal{C}, \mathcal{T} \models D \iff D' \).

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

Example 14
Let us consider the following program \( P \) :

\[
P_1 = \{ \quad r_1 \@ p(X) \iff q(X), s(X) \quad \\
\quad r_2 \@ t(a) \iff r(b) \quad \\
\quad r_3 \@ q(Y) \iff t(Y) \quad \\
\quad r_4 \@ s(a) \setminus q(a) \iff r(b) \quad \}
\]

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

\[
P_1 = (P \setminus \{r_1\}) \cup \text{Unf}_P(r_1),
\]

where \( P_1 \) is the program

\[
P_1 = \{ \quad r_1 \@ p(X) \iff s(X), t(Y), X = Y \quad \\
\quad r_2 \@ t(a) \iff r(b) \quad \\
\quad r_3 \@ q(Y) \iff t(Y) \quad \\
\quad r_4 \@ s(a) \setminus q(a) \iff r(b) \quad \}.
\]
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 given 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

$$P' = (P \setminus \{cl\}) \cup \text{Unf}_P(cl).$$

If $P$ is normally terminating then $P'$ is also 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:

$$P = \{ r_1@p(X) \Leftarrow q(X) \\
r_2@q(a) \Leftarrow p(a) \\
r_3@q(Y) \Leftarrow r(Y) \}$$

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 $P'$:

$$P' = \{ r_1@p(X) \Leftarrow X = Y, r(Y) \\
r_2@q(a) \Leftarrow p(a) \\
r_3@q(Y) \Leftarrow r(Y) \}$$

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 $r_1@p(X) \Leftarrow q(X)$ and $r_2@q(a) \Leftarrow p(a)$ in $P$ can be applied infinitely many times. Hence, we have non-(normal)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

$$P' = (P \setminus \{cl\}) \cup \text{Unf}_P(cl).$$

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

Analogous to Theorem 1, by using Proposition 1 we can prove that \( \mathcal{A}'_{p}(G) = \mathcal{A}'_{p'}(G) \), where

\[
P'' = P \cup Unf_{P}(cl),
\]

for any arbitrary goal \( G \).

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

\[ \mathcal{A}'_{p'}(G) = \mathcal{A}'_{p''}(G). \]

We prove the two inclusions separately.

\( (\mathcal{A}'_{p'}(G) \subseteq \mathcal{A}'_{p''}(G)) \) The proof is the same as in the case \( \mathcal{A}'_{p'}(G) \subseteq \mathcal{A}'_{p''}(G) \) of Theorem 1 and hence it is omitted.

\( (\mathcal{A}'_{p''}(G) \subseteq \mathcal{A}'_{p'}(G)) \) The proof is by contradiction. Assume that there exists \( Q \in \mathcal{A}'_{p''}(G) \setminus \mathcal{A}'_{p'}(G) \). Since, from the proof of Proposition 4 we can conclude that \( P'' \) is normally terminating and confluent, we have that \( \mathcal{A}'_{p'}(G) \) is a singleton. Moreover, since by the previous point \( \mathcal{A}'_{p'}(G) \subseteq \mathcal{A}'_{p''}(G) \), we have that \( \mathcal{A}'_{p'}(G) = \emptyset \). This means that each normal derivation in \( P' \) is either 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 \( P'' \) that terminate with a failed configuration. Then by Lemma 3, and since \( Q \in \mathcal{A}'_{p''}(G) \), 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 also hold 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 \( CT \models D \leftrightarrow D' \).

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 that 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 with respect to a specific goal, analogous 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 that involves program transformation for CHR is Sarna-Starosta and Schrijvers (2009).

Both general and goal-specific approaches are important 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), which deals with the transformation of Concurrent Constraint Programming (CCP) and Ueda and Furukawa (1988), which considers Guarded Horn Clauses (GHC). The results in these papers are not directly applicable to CHR because neither CCP nor GHC allows 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 noting 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, different from the conditions given in Tacchella et al. (2007) and Tacchella (2008), the conditions defined in Section 5 allow us to perform rule replacement also when rules with multiple heads are used for unfolding a given rule. This is a major improvement, since CHR rules naturally have multiple heads.

The results obtained in the current paper can be considered as the 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 also take into account 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 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 that are used often and leave those that 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 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 that one would need to decide whether and in which sequence to unfold CHR rules could be obtained experimentally, e.g., by profiling, or formally via probabilistic program analysis. One could see this as a kind of speculative unfolding.

Appendix A Proofs

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

Appendix A.1 Equivalence of the operational semantics

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

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

Definition 23 (Configuration equivalence)
Let $\sigma = (H_1, C, H_2, D, T)_n \in \text{Conf}_i$ be a configuration in the transition system $\omega_i$ and let $\sigma' = ((K, C), D, T')_m \in \text{Conf}'_i$ be a configuration in the transition system $\omega'_i$.

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

1. there exist $K_1$ and $K_2$, such that $K = K_1 \sqcup K_2$, $H_1 = \text{chr}(K_1)$ 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 $\rho$ s.t. $H_2 \rho = K_2$ and $T \rho = T'$.

Condition (1) grants that $\sigma$ and $\sigma'$ have equal CHR constraints, while condition (2) ensures that no propagation rule is applied to constraints in $\sigma'$ corresponding to constrains in $\sigma$ 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 $\sigma$ and the ones associated with them in $\sigma'$ 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 $Ann(P)$ be respectively a CHR program and its annotated version. Moreover, let $\sigma \in Confi$ and let $\sigma' \in Confi'$ such that $\sigma \approx \sigma'$. Then the following holds:

- There exists a derivation $\delta = \sigma \rightarrow^*_o \sigma_1$ in $P$ if and only if there exists a derivation $\delta' = \sigma' \rightarrow^*_o \sigma'_1$ in $Ann(P)$ such $\sigma_1 \approx \sigma'_1$

- The number of $\text{Solve}$ ($\text{Apply}$) transition steps in $\delta$ and the number of $\text{Solve}'$ ($\text{Apply}'$) transition steps in $\delta'$ 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 $\text{Solve}$ ($\text{Apply}$) transition step in $\delta$ if and only if there exists a $\text{Solve}'$ ($\text{Apply}'$) transition step in $\delta'$.

Then the proof follows by a straightforward inductive argument.

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

$$K = K_1 \cup K_2, \quad H_1 = \text{chr}(K_1), \quad \text{chr}(H_2) = \text{chr}(K_2), \quad H_2\rho = K_2$$

(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 $C = C' \cup \{c\}$. By the definitions of the two transition systems,

$$\sigma \rightarrow_{\text{Solve}}^{\omega_t} \langle (H_1, C'), H_2, D \land c, T \rangle_n \quad \text{and} \quad \sigma' \rightarrow_{\text{Solve}'}^{\omega_t} \langle (K, C'), D \land c, T' \rangle_m.$$

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

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

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

Let $H'_2 = H_2 \cup \{h\#n\}$. By (A1) and since $H_1 = H'_1 \cup \{h\}$, 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 $\rho$).

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

Moreover, by definition of $\approx$, for each $l \in id(K_1)$, $l$ does not occur in $T'$. Therefore, since by construction $K'_1 \subseteq K_1$, we have that for each $l \in id(K'_1)$, $l$ 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 $\rho'$ 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, $\rho'$ 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, $f \notin id(K_2) = id(H_2\rho)$, we have that $H_2\rho' = H_2\rho\{n'/f, f/n\} = H_2\rho\{n'/f\}$. Moreover, since by definition $n \notin id(H_2)$ and $n' = \rho(n)$, we have that $H_2\rho\{n'/f\} = H_2\rho$. By the previous observations, we have that

$$H_2\rho' = H_2\rho \cup \{h\#n\}\{n'/f\} = K_2'. $$

Finally, we prove that $T\rho' = T'$. Since by the definition of configurations in $Conf$, $n$ 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 $f \in id(K_1)$, we have that $f$ does not occur in $T'$. Therefore, $T'\{f/n'\} = T'$ and then the thesis.

**Apply and Apply’**: Let $cl_r = r@F'\backslash F'' \iff D_1 | B, C_1 \in P$ and let $cl'_r = r@F'\backslash F'' \iff D_1 | \tilde{B}, C_1 \in Ann(P)$ be its annotated version, where $\tilde{B} = (I(B))$. The latter can be applied to the considered configuration $\sigma' = \langle(K, C), D, T'\rangle_n$. In particular, $F', F''$ match respectively with $P_1$ and $P_2$ such that $P_1 \cup P_2 \subseteq K$. Without loss of generality, by using a suitable number of Introduce steps, we can assume that $r@F'\backslash F'' \iff D_1 | B, C_1 \in P$ 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 $Q_1 \cup Q_2 \subseteq H_2$, $Q_1\rho = P_r$, and $F', F''$ match respectively with $Q_1$ and $Q_2$.

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

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

- $r@id(P_1, P_2) \notin T'$ if and only if $r@id(Q_1, Q_2) \notin T$ and
- $\langle \sigma, t \rangle \models D \implies \exists cl_t(((F', F'') = \overline{chr}(P_1, P_2)) \land D_1)$ if and only if $\langle \sigma, t \rangle \models D \implies \exists cl_t(((F', F'') = \overline{chr}(Q_1, Q_2)) \land D_1)$.

Therefore, by the definitions of **Apply** and **Apply’**

$$\sigma \xrightarrow{\text{Apply}}_{o_1} \langle \{H_1, C\} \cup \{B, C_1\}, (Q_1, Q_3), ((F', F'') = \overline{chr}(Q_1, Q_2)) \land D_1 \land D, T_1\rangle_n$$

if and only if

$$\sigma' \xrightarrow{\text{Apply’}}_{o_1'} \langle \{K_1, P_1, P_3, C, B', C_1\}, ((F', F'') = \overline{chr}(P_1, P_2)) \land D_1 \land D, T'_1\rangle_o$$

where

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

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

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 $(K_1, P_1, P_3, B') = K'_1 \cup K'_2$, $H_1 \cup B = chr(K'_1)$ and $chr(Q_1, Q_3) = chr(K'_2)$. 
2. Since for each \( l \in id(K_1) \), \( l \) does not occur in \( T' \), \( P_1 \subseteq K_2 \), and by definition of \textbf{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 \( \sigma_1 \approx \sigma'_1 \) and then the thesis. \( \Box \)

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 \( G \),

\[ \mathcal{2} \mathcal{A}_P(G) = \mathcal{2} \mathcal{A}'_{Ann(P)}(G) \]

holds.

**Proof**
By definition of \( \mathcal{2} \mathcal{A} \) and of \( \mathcal{2} \mathcal{A}' \), the initial configurations of the two transition systems are equivalent. Then the proof follows by Lemma 1. \( \Box \)

**Appendix A.2 Correctness of the unfolding**

We now prove the correctness of our unfolding definition.

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

**Proposition 5**
Let \( P \) be CHR program, and let \( P' \) be an annotated CHR program. Then

\[ \mathcal{2} \mathcal{A}_P(G) = \{ \exists Fv(G) (\text{chr}(K) \land d) \mid E \not\models d \leftrightarrow \text{false}, \delta = \langle G, \emptyset, \text{true}, \emptyset \rangle \vdash \bullet \langle \emptyset, K, d, T \rangle \not\vdash \omega \}
\]

and

\[ \mathcal{2} \mathcal{A}'_{P'}(G) = \{ \exists Fv(G) (\text{chr}(K) \land d) \mid E \not\models d \leftrightarrow \text{false}, \delta = \langle I(G), \text{true}, \emptyset \rangle \vdash \bullet \langle K, d, T \rangle \not\vdash \omega \}
\]

The next proposition essentially shows the correctness of unfolding with respect to a derivation step. We first define an equivalence between configurations in \( \text{Conf}' \).

**Definition 24 (Configuration Equivalence)**
Let \( \sigma = \langle G, D, T \rangle \) and \( \sigma' = \langle G', D', T' \rangle \) be configurations in \( \text{Conf}' \). \( \sigma \) and \( \sigma' \) are equivalent and we write \( \sigma \approx \sigma' \) if one of the following facts hold:

- \( \sigma \) and \( \sigma' \) are both failed configurations, or
- \( G = G', E \models D \leftrightarrow D' \) and \( \text{clean}(G, T) = \text{clean}(G', T') \).
**Proposition 6**

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

**Proof**

Assume that

\[
\sigma \rightarrow^{cl'_r} \sigma' \rightarrow^{Solve'} \sigma'_f \rightarrow^{cl_e} \sigma_e \rightarrow^{Solve'} \sigma^f.
\]

The labeled arrow \( \rightarrow^{Solve'} \) means that only \textbf{Solve} transition steps are applied.

Moreover:

- If \( \sigma^f \) has the form \( \langle G, \text{false} \rangle, T \rangle \) then the derivation between the parenthesis is not present and \( \sigma^f = \sigma'^f \).
- The derivation between the parenthesis is present and \( \sigma^f = \sigma'_f \), otherwise.

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

\[
\mathcal{E}, \mathcal{F} \models (D_r \wedge C_r) \rightarrow \text{chr}(S_1,S_2) = (S'_1,S'_2) \theta.
\]

(A2)

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

\[
r \mathbin{id}(S_1,S_2) \notin T_r, V \subseteq D_v, V = \{ c \mid \mathcal{E}, \mathcal{F} \models (D_r \wedge C_r) \rightarrow c \theta \}, D'_v = D_v \setminus V, Fv(D'_v) \cap Fv((S'_1,S'_2) \theta) \subseteq Fv(H'_1,H'_2), \text{the constraint } (D_r, \langle D'_v \theta \rangle) \text{ is satisfiable and}
\]

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

By the previous observations, we have that

\[
\mathcal{E}, \mathcal{F} \models (D_r \wedge C_r) \rightarrow V \theta,
\]

(A3)

and therefore \( \mathcal{E}, \mathcal{F} \models V \theta \iff \exists_{Fv(D_r \wedge C_r)} V \theta \). Then, without loss of generality, we can assume that

\[
Fv(V \theta) \subseteq Fv(cl_r).
\]

(A4)

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

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

(A5)

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

\[
Fv(D'_v \theta) \subseteq Fv(H'_1,H'_2) \cup Fv(cl_r).
\]

(A6)
Let us consider the application of the rule $cl'$ to $\sigma$. By definition of the transition $\text{Apply}'$, we have that

$$\mathcal{E}. \mathcal{T} \models C \rightarrow \exists_{cl'}((\text{chr}(H_1, H_2) = (H_1', H_2')) \land D_r \land (D'_r \theta)) \quad \text{(A7)}$$

and

$$\sigma_r' = ((Q, C_r, C_v, \text{chr}(S_1, S_2) = (S_1', S_2')), (D, T_3))_{j+m_1},$$

where

- $Q = (H_1, H_3, Q_1)$,
- $\mathcal{E}. \mathcal{T} \models D \leftrightarrow (\text{chr}(H_1, H_2) = (H_1', H_2') \land D_r \land (D'_r \theta) \land C)$,
- $\text{inst}((K, S_1, P_1), T_r', j) = (Q_1, T'_r, j + m_1)$ and $T_3 = T \cup T'_r \cup \{r \cdot id(H_1, H_2)\}$.

Therefore, by definition $\sigma'_r = ((Q, C'_r, T_3))_{j+m_1}$, where

$$\mathcal{E}. \mathcal{T} \models C'_r \leftrightarrow (C_r \land C_v \land \text{chr}(S_1, S_2) = (S_1', S_2') \land D).$$

Let us now consider the application of $cl_r$ to $\sigma$ and then of $cl_v$ to the $\sigma'_r$ obtained from the previous application. Since by construction $Fv((\text{chr}(H_1, H_2) = (H_1', H_2')) \land D_r) \cap Fv(cl'_v) \subseteq Fv(cl_r)$ and by (A7), we have

$$\mathcal{E}. \mathcal{T} \models C \rightarrow \exists_{cl}((\text{chr}(H_1, H_2) = (H_1', H_2')) \land D_r).$$

Therefore, by the definition of the transition $\text{Apply}'$, we have

$$\sigma_r = ((Q_2, C_r, \text{chr}(H_1, H_2) = (H_1', H_2') \land D_r \land C, T_4))_{j+m},$$

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 $T_4 = T \cup T_2 \cup \{r \cdot id(H_1, H_2)\}$.

Therefore, by definition $\sigma'_r = ((Q_2, C'_r, T_4))_{j+m}$, where

$$\mathcal{E}. \mathcal{T} \models C'_r \leftrightarrow C_r \land \text{chr}(H_1, H_2) = (H_1', H_2') \land D_r \land C. \quad \text{(A8)}$$

Now we have two possibilities:

($C'_r = \text{false}$). In this case, by construction, we have that $C'_r = \text{false}$. Therefore, $\sigma'_r \simeq \sigma'_r$ and then the thesis.

($C'_r \neq \text{false}$). By (A8), (A3) and (A2), we have that

$$\mathcal{E}. \mathcal{T} \models C'_r \rightarrow \text{chr}(S_1, S_2) = (S_1', S_2') \theta \land V \theta.$$

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

$$\mathcal{E}. \mathcal{T} \models C'_r \rightarrow \exists_{H_1', H_2', cl}((\text{chr}(H_1, H_2) = (H_1', H_2') \land (D'_r \theta) \land \text{chr}(H_1, H_2) = (H_1', H_2'))$$

and then $\mathcal{E}. \mathcal{T} \models C'_r \rightarrow \exists_{cl}(D'_r \theta)$. Therefore, by (A4), (A5), and since the rules are renamed apart,

$$\mathcal{E}. \mathcal{T} \models C'_r \rightarrow \exists_{cl}(\text{chr}(S_1, S_2) = (S_1', S_2') \theta \land V \theta \land D'_r \theta).$$

Then by definition of $D_r$ and since $\text{dom}(\theta) \subseteq Fv(S_1', S_2')$, we have that $\mathcal{E}. \mathcal{T} \models C'_r \rightarrow \exists_{cl}(\text{chr}(S_1, S_2) = (S_1', S_2') \land D_r \theta)$. 

Unfolding for CHR programs 299
Therefore, since \( \text{dom}(\theta) \subseteq \text{Fv}(S'_1, S'_2) \subseteq \text{Fv}(c_\nu) \),

\[
\forall \mathcal{F} \models C^f \rightarrow \exists c_\nu (\text{chr}(S_1, S_2) = (S'_1, S'_2) \land D_\nu).
\]

Then \( \sigma^f \) is such that we can use the transition \textbf{Apply} with the rule \( c_\nu \) obtaining the new configuration

\[
\sigma_v = \langle (Q_3, C_v), D', T_5 \rangle_m,
\]

where

- \( Q_3 = (H_1, H_3, K'', S''_1, P_2) \)
- \( \forall \mathcal{F} \models D' \leftarrow (\text{chr}(S_1, S_2) = (S'_1, S'_2) \land D_\nu \land C_v \land \text{chr}(H_1, H_2) = (H'_1, H'_2) \land D_\nu \land C) \),
- \( \text{inst}(P, T_r, j + m) = (P_2, T'_r, m_1) \) and \( T_5 = T_4 \cup T'_r \cup \{v@id(S''_1, S''_2)\} \).

Finally, by definition, we have that \( \sigma^f_v = \langle Q_3, C^f_v, T_5 \rangle_m \), where

\[
\forall \mathcal{F} \models C^f_v \leftarrow C_v \land D'.
\]

By definition of \( D \) and \( D' \), we have that \( \forall \mathcal{F} \models C^f_v \leftarrow C^f_v \).

If \( C^f_v = \text{false} \) 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

\[
\text{clean}(Q, T_3) = \text{clean}(Q, T_5).
\]

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

\[
T_3 = T \cup T'_r \cup \{r@id(H_1, H_2)\}
= T \cup \text{inst}'(\text{clean}(K, S_1), T_r, j) \cup \text{inst}'(T_r, j + m) \cup \text{inst}'(\{v@id(S_1, S_2)\}, j) \cup \{r@id(H_1, H_2)\}.
\]

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

\[
T_5 = T_4 \cup T'_r \cup \{v@id(S''_1, S''_2)\}
= T \cup \text{inst}'(T_r, j) \cup \{r@id(H_1, H_2)\} \cup \text{inst}'(T_r, j + m) \cup \{v@id(S''_1, S''_2)\}.
\]

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 \( \text{inst}'(\{v@id(S_1, S_2)\}, j) = \{v@id(S''_1, S''_2)\} \). 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 \((K, S_i)\) and by adding the natural \(j + m\) to each identifier in \(P\). Then
\[\text{clean}(Q, \text{inst}(\text{clean}(K, S_i), T_r), j)) = \text{clean}(Q, \text{inst}(T_r, j))\]
and then the thesis holds.

\[\square\]

Hence, we obtain the correctness result.

**Proposition 2**

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

**Proof**

We prove the two inclusions separately.

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

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

\[\delta = \langle I(G), \text{true}, \emptyset \rangle_m \xrightarrow{o_i} \langle K, d, T \rangle_n \xrightarrow{o_i} \langle K, d', T' \rangle_n\]

in \(P\), such that \(Q = \exists_{Fh(G)}(\text{chr}(K) \land d)\). Since \(P \subseteq P'\), we have that there exists the derivation \(\langle I(G), \text{true}, \emptyset \rangle_m \xrightarrow{o_i} \langle K, d, T \rangle_n \xrightarrow{o_i} \langle K, d', T' \rangle_n\) in \(P'\). Moreover, since \(P' = P \cup \{cl'_r\}\) and by hypothesis \(Q \notin \mathcal{A}'P'(G)\), we have that there exists a derivation step \(\langle K, d, T \rangle_n \xrightarrow{o_i} \langle K_1, d_1, T_1 \rangle_n\) by using the rule \(cl'_r\). Then by the definition of unfolding there exists a derivation step \(\langle K, d, T \rangle_n \xrightarrow{o_i} \langle K_2, d_2, T_2 \rangle_n\) in \(P\), by using the rule \(cl_e\) and then we have a contradiction. \(\square\)

**Appendix A.3 Safe replacement**

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

**Proposition 7**

Let \(cl_r, cl_e\) be two annotated CHR rules such that the following holds:

- \(cl_r\) is of the form \(r \ast H_1 \setminus H_2 \Rightarrow D_r \mid K_r; T_r\),
- \(cl'_r \in \text{Unf}_{\{cl_r\}}(cl)\) is of the form \(r \ast H_1 \setminus H_2 \Rightarrow D'_r \mid K'_r; T'_r\), with \(\not\in \mathcal{T} \models D_r \leftrightarrow D'_r\)

and it is obtained by unfolding the identified atoms \(A \subseteq K_r\).

Moreover, let \(\sigma\) be a generic built-in free configuration such that we can construct a derivation \(\delta\) from \(\sigma\), where

- \(\delta\) uses at the most the rules \(cl_r\) and \(cl_e\) in the order,
- a built-in free configuration \(\sigma^f\) can be obtained, and
- if \(cl_e\) is used, then \(cl_e\) rewrites the atoms \(A'\) such that \(\text{chr}(A) = \text{chr}(A')\).

Then we can use the transition \(\text{Apply}'\) with the rule \(cl'_r\) obtaining the configuration \(\sigma_{\cdot'}\) and then the built-in free configuration \(\sigma'_{\cdot'}\) such that \(\sigma'_{\cdot'} \simeq \sigma^f\).
Proof
Assume that
\[ \sigma \xrightarrow{cl_r} \sigma_r \xrightarrow{Solve^C} \sigma_f (\xrightarrow{cl_v} \sigma_v \xrightarrow{Solve^C} \sigma_f^v) \]

The labeled arrow \( \xrightarrow{Solve^C} \) means that only \texttt{Solve} transition steps are applied. Moreover,
- if \( \sigma_f^v \) has the form \( \langle G, \texttt{false}, T \rangle \) then the derivation between the parenthesis is not present and \( \sigma_f = \sigma_f^v \),
- the derivation between the parenthesis is present and \( \sigma_f = \sigma_f^v \), 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 \( r@H_1 \setminus H_2 \Rightarrow D_r | K, A, C_r; T_r \) and \( v@H_1' \setminus H_2' \Rightarrow D_v | P, C_v; T_v \), respectively, \( A = A_1 \sqcup A_2 \), \( C_r \) is the conjunction of all the built-in constraints in the body of \( cl_r \) and \( \theta \) is a substitution such that \( \text{dom}(\theta) \subseteq \text{Fv}(H_1', H_2') \) and

\[ \mathcal{C}.\mathcal{T} \models (D_r \land C_r) \rightarrow \text{chr}(A_1, A_2) = (H_1', H_2')\theta. \tag{A9} \]

Furthermore, let \( m \) be the greatest identifier that appears in the rule \( cl_r \), and let \( (P_1, T_1, m_1) = \text{inst}(P, T, m) \).

Then the unfolded rule \( cl_r' \) is:
\[ r@H_1 \setminus H_2 \Rightarrow D_r'(D_v'\theta) | K, A_1, P_1, C_r, C_r, \text{chr}(A_1, A_2) = (H_1', H_2'); T_r' \]
where \( v@id(A_1, A_2) \notin T_r, \ V = \{ d \in D_v \mid \mathcal{C}.\mathcal{T} \models (D_r \land C_r) \rightarrow \theta \}, \ D_v' = D_v \setminus V, \ Fv(D_v'\theta) \cap Fv(k')\theta \subseteq \text{Fv}(H_1', H_2') \), 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 \( T_r' = \text{clean}((K, A_1), T_r) \cup T_1 \).

Since by hypothesis, \( \mathcal{C}.\mathcal{T} \models (D_r, (D_v'\theta)) \leftrightarrow D_r \), we have that
\[ \mathcal{C}.\mathcal{T} \models (D_r \land C_r) \rightarrow D_r\theta \text{ and } D_v'\theta = \emptyset. \tag{A10} \]

Let us now consider the application of the rule \( cl_r \) to \( \sigma \). By definition of the \texttt{Apply}' transition step, we have
\[ \mathcal{C}.\mathcal{T} \models C \rightarrow \exists_{cl_r} (\text{chr}(F_1, F_2) = (H_1, H_2) \land D_r) \tag{A11} \]
and
\[ \sigma_r = \langle (Q_2, C_r), \text{chr}(F_1, F_2) = (H_1, H_2) \land D_r \land C, T_4 \rangle_{j+m}, \]
where \( Q_2 = (F_1, F_3, K', A') \), \( ((K', A'), T_2, j+m) = \text{inst}((K, A), T_r, j) \) and \( T_4 = T \cup T_2 \cup \{ r@id(F_1, F_2) \} \).

Therefore, by definition
\[ \sigma_f^v = \langle Q_2, C_r', T_4 \rangle_{j+m}, \]
where
\[ \mathcal{C}.\mathcal{T} \models C_r' \leftrightarrow C_r \land \text{chr}(F_1, F_2) = (H_1, H_2) \land D_r \land C. \tag{A12} \]
Let us now apply the rule \( c l' \) to \( \sigma \). By (A11), (A10) and by the definition of the \textit{Apply} transition step, we have that

\[
\sigma' = \langle (Q, C_r, C_v, chr(A_1, A_2) = (H_1', H_2') \rangle, D, T_3 \rangle_{j+m_1},
\]

where

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

Therefore, by definition

\[
\sigma'_v = \langle Q, C'_r, T_3 \rangle_{j+m_1},
\]

where

\[
\mathcal{E} \models C'_r \iff C_r \land C_v \land chr(A_1, A_2) = (H_1', H_2') \land D.
\]

Now we consider the two previously obtained configurations \( \sigma'_v \) and \( \sigma'_r \). Since by hypothesis \( \sigma'_v \) is a non-failed configuration, we have that \( C'_r \neq \text{false} \).

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 \( cl_r \), \( dom(\theta) \subseteq Fv(H_1', H_2') \subseteq Fv(cl_r) \), by (A12), (A9), and (A10), we have that

\[
\mathcal{E} \models C'_r \iff ((chr(A_1, A_2) = (H_1', H_2')) \land D_r) \theta
\]

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

\[
\mathcal{E} \models C'_r \iff \exists_{cl_r}(((chr(A_1, A_2) = (H_1', H_2')) \land D_r).
\]

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

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

Finally, by definition, we have that \( \sigma'_v = \langle Q_3, C'_v, T_5 \rangle_{m_1} \), where

\[
\mathcal{E} \models C'_v \iff (C_v \land D').
\]

If \( C'_v = \text{false} \) 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. \( \square \)

\textbf{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) \( cl = r @ H \backslash H_2 \iff D \mid A; T \) where \((H_1, H_2) = (h_1, \ldots, h_n)\),
(b) there exists \((K_1, K_2) = (k_1, \ldots, k_n) \subseteq F\) such that \( r @ id(K_1, K_2) \notin T\) and \( \mathcal{C} \mathcal{F} \models c \rightarrow \exists_{cl}(\text{chr}(K_1, K_2) = (H_1, H_2)) \land D\),
(c) there exist \( l \in \{1, \ldots, n\} \) and \((K'_1, K'_2) = (k'_1, \ldots, k'_n) \subseteq F\) such that \( k_l = k'_l\), \( r @ id(K'_1, K'_2) \notin T\) and \( \mathcal{C} \mathcal{F} \models c \land \text{chr}(k_l) = h_l \rightarrow (\text{chr}(K'_1, K'_2) = (H_1, H_2))\),
(d) \( \sigma_0 \rightarrow_{o' \sigma} \sigma \) is an \textbf{Apply}' transition step which uses the clause \( cl\), rewrites the atoms \((K_1, K_2)\) such that \( \sigma = \langle ((F \setminus K_1) \cup A'), C, T'' \rangle_{m'}\), where \( C \) is the constraint \( (\text{chr}(K_1, K_2) = (H_1, H_2)) \land D \land c \).

Then there exists an \textbf{Apply}' transition step \( \sigma_0 \rightarrow_{o' \sigma} \sigma' \) which uses the clause \( cl\), rewrites the atoms \((K'_1, K'_2)\) and such that \( \sigma' = \langle ((F \setminus K'_1) \cup A'), C', T'' \rangle_{m'}\), where \( C' \) is the constraint \( (\text{chr}(K'_1, K'_2) = (H_1, H_2)) \land D \land c \) and

1. \( \mathcal{C} \mathcal{F} \models (\text{chr}(F \setminus K_1) \land A') \land C \leftrightarrow (\text{chr}(F \setminus K'_1) \land A') \land C'\),
2. \( T'' = (T' \setminus [r @ id(K_1, K_2)]) \cup r @ id(K'_1, K'_2)\).

\textbf{Proof}
First of all, by definition of \textbf{Apply}' transition step and since, by hypothesis (e), \( r @ id(K'_1, K'_2) \notin T\), we have to prove that

\[ \mathcal{C} \mathcal{F} \models c \rightarrow \exists_{cl}(\text{chr}(K'_1, K'_2) = (H_1, H_2)) \land D. \]

By hypothesis (b) and since \( Fv(c) \cap Fv(cl) = 0\), we have that \( \mathcal{C} \mathcal{F} \models c \rightarrow \exists_{cl}(c \land (\text{chr}(k_l) = h_l)) \land D\). Hence, the thesis follows hypothesis (c).

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

\[ \mathcal{C} \mathcal{F} \models c \rightarrow (\text{chr}(K_1, K_2) = (H_1, H_2)) \theta. \]  

(A13)

By hypothesis (c), and since \( \text{dom}(\theta) \cap Fv(c, K_1, K_2) = 0\), we have that

\[ \mathcal{C} \mathcal{F} \models (c \land (\text{chr}(k_l) = h_l)) \theta \rightarrow (\text{chr}(K'_1, K'_2) = (H_1, H_2)) \theta \]

and by (A13), \( \mathcal{C} \mathcal{F} \models c \rightarrow (\text{chr}(k_l) = h_l) \theta \).

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

The proof of (2) is obvious by the definition of \textbf{Apply}' transition step. \qed

\textbf{Proposition 9}
Let \( \sigma_0 = \langle F, c, T \rangle_{m} \) be a built-in configuration such that there exists a normal terminating derivation \( \delta \) starting from \( \sigma \) which ends in a configuration \( \sigma' \). Assume that \( \delta \) uses an annotated CHR rule \( cl\) such that the following holds:

(a) \( cl = r @ H \backslash H_2 \iff D \mid A; T \)
(b) there exists \((K_1, K_2) \subseteq F\) such that \( cl\) rewrites the atoms \((K_1, K_2)\) in \( \delta \) and \( \mathcal{C} \mathcal{F} \models c \rightarrow \exists_{cl}(\text{chr}(K_1, K_2) = (H_1, H_2)) \land D \)

Then there exists a normal terminating derivation \( \delta' \) starting from \( \sigma_0 \) such that

- \( \delta' \) uses at most the same clauses of \( \delta \) and uses the rule \( cl\) in the first \textbf{Apply}' transition step, in order to rewrite the atoms \((K_1, K_2)\),
- \( \delta' \) ends in a configuration \( \sigma' \) such that \( \sigma \simeq \sigma' \).
Unfolding for CHR programs

Proof

The proof is obvious by the definition of derivation.

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

\[
P' = (P \setminus \{cl\}) \cup \text{Unf}_P(cl).
\]

Then \( 2\mathcal{A}'_p(G) = 2\mathcal{A}'_{\text{pr}}(G) \) for any arbitrary goal \( G \).

Proof

By using a straightforward inductive argument and by Proposition 2, we have

\[
2\mathcal{A}'_p(G) = 2\mathcal{A}'_{\text{pr}}(G),
\]

where

\[
P'' = P \cup \text{Unf}_P(cl)
\]

for any arbitrary goal \( G \).

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

\[
2\mathcal{A}'_p(G) = 2\mathcal{A}'_{\text{pr}}(G).
\]

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

\([2\mathcal{A}'_p(G) \subseteq 2\mathcal{A}'_{\text{pr}}(G)]\) The proof is by contradiction. Assume that there exists \( Q \in 2\mathcal{A}'_p(G) \setminus 2\mathcal{A}'_{\text{pr}}(G) \). By definition there exists a derivation

\[
\delta = \langle I(G), \text{true}, \emptyset \rangle_m \rightarrow^*_{o_i} \langle K, d, T \rangle_n \not\rightarrow^*_{o_i}
\]

in \( P' \) such that \( Q = \exists_{F_i(G)}(\text{chr}(K) \land d) \). Since \( P' \subseteq P'' \), we have that there exists the derivation

\[
\langle I(G), \text{true}, \emptyset \rangle_m \rightarrow^*_{o_i} \langle K, d, T \rangle_n
\]

in \( P'' \). Moreover, since \( P'' = P' \cup \{cl\} \) and \( Q \notin 2\mathcal{A}'_{\text{pr}}(G) \), we have that there exists a derivation step \( \langle K, d, T \rangle_n \rightarrow^*_{o_i} \langle K_1, d_1, T_1 \rangle_m \) by using the rule \( cl \).

Since \( cl \) can be safely replaced in \( P \), we have that there exists an unfolded rule \( cl' \in \text{Unf}_P(cl) \) such that \( cl' \) is of the form

\[
r \vdash H_1 \setminus H_2 \Leftrightarrow D' \mid A'; T',
\]

\( \mathcal{C} \vdash D \leftarrow D' \) and by construction \( cl' \in P' \).

Then there exists a derivation step \( \langle K, d, T \rangle_n \rightarrow^*_{o_i} \langle K_2, d_2, T_2 \rangle_n \) in \( P' \) (by using the rule \( cl' \)) and then we have a contradiction.

\([2\mathcal{A}'_{\text{pr}}(G) \subseteq 2\mathcal{A}'_p(G)]\) First of all observe that by Proposition 5, \( 2\mathcal{A}'_{\text{pr}}(G) \) can be calculated by considering only non-failed normal terminating derivations. Then for each non-failed normal terminating derivation \( \delta \) in \( P'' \), which uses the rule \( cl \) after the application of \( cl \), we obtain the configuration \( \sigma_1 \) and then a non-failed built-in free configuration \( \sigma_1' \). Now let \( C \) be the built-in constraint store of \( \sigma_1' \).
Since by hypothesis $cl$ can be safely replaced in $P$, following Definition 11, we have that 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 $\delta$ by using a rule $cl'$ in $P$. Therefore, without loss of generality, we can assume that

$$\delta = \langle I(G), \text{true}, \emptyset \rangle_{m} \rightarrow_{o_{i}'} \sigma \rightarrow_{o_{i}'} \sigma_{1} \rightarrow_{o_{i}'} \sigma_{1}' \rightarrow_{o_{i}'} \sigma_{2} \rightarrow_{o_{i}'} \sigma_{3} \rightarrow_{o_{i}'} \sigma_{4} \rightarrow_{*} \sigma'$$

where the transition step $s_{1} = \sigma \rightarrow_{o_{i}'} \sigma_{1}$ is the first Apply' transition step which uses the clause $cl$ and $s_{2} = \sigma_{2} \rightarrow_{o_{i}'} \sigma_{3}$ 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 $P$, 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.

\[\square\]

**Appendix A.4 Termination and confluence**

We first prove the correctness of unfolding with respect to termination.

**Proposition 3 (Normal Termination)**

Let $P$ be a CHR program and let $P_{0}, \ldots, P_{n}$ 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 Propositions 6 and 7 and by using a straightforward inductive argument, we have that for each $i = 0, \ldots, n - 1$, $P_{i}$ satisfies normal termination if and only if $P_{i+1}$ satisfies the normal termination too and then the thesis.

The following lemma relates \(\approx, \simeq\) and \(\equiv_{V}\) equivalences.

**Lemma 2**

Let $\sigma, \sigma'$ be final configurations in $Conf_{f}$, $\sigma_{1}, \sigma_{2}, \sigma_{1}', \sigma_{2}' \in Conf'_{f}$, 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 $\sigma$ is a final configuration in $Conf_{f}$, then $\sigma$ has the form $(G,S,\text{false},T)_{n}$ or it has the form $(\emptyset,S,c,T)_{n}$.

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 variable $V$. \[\square\]
**Theorem 2 (Confluence)**
Let \( P \) be a CHR program and let \( P_0, \ldots, P_n \) be an NRU-sequence starting from \( P_0 = \text{Ann}(P) \). \( P \) 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 \( \text{Ann}(P) \) is confluent. Moreover, we prove that for each \( i = 0, \ldots, n-1 \), \( P_i \) is confluent if and only if \( P_{i+1} = (P_i \setminus \{ \text{cl}^l \}) \cup \text{Unf}_P(\text{cl}^l) \) is confluent. 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 satisfy confluence. By definition, there exists a state \( \sigma = \langle (K, D), C, T \rangle_0 \) and two derivations \( \sigma \xrightarrow{o_i} \sigma_1 \) and \( \sigma \xrightarrow{o_i} \sigma_2 \) in \( P_{i+1} \) such that there are no two derivations \( \sigma_1 \xrightarrow{r_i} \sigma'_1 \) and \( \sigma_2 \xrightarrow{r_i} \sigma'_2 \) in \( P_{i+1} \) where \( \sigma'_1 \equiv_{Fv(\sigma)} \sigma'_2 \). Without loss of generality, we can assume that \( \sigma_1 \) and \( \sigma_2 \) are built-in free states. Therefore, by Proposition 6, there exist two derivations \( \sigma \xrightarrow{o_i} \sigma_3 \) and \( \sigma \xrightarrow{o_i} \sigma_4 \) 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 \( \delta = \sigma_3 \xrightarrow{o_i} \sigma'_3 \) and \( \delta' = \sigma_4 \xrightarrow{o_i} \sigma'_4 \) in \( P_i \) such that \( \sigma'_3 \equiv_{Fv(\sigma)} \sigma'_4 \). Without loss of generality, we can assume that \( \sigma'_3 \) and \( \sigma'_4 \) are built-in free configurations. Analogous to Theorem 1, since by hypothesis \( cl^l \) can be safely replaced in \( P_i \) and by using Proposition 7, we can construct two new derivations \( \gamma = \sigma_1 \xrightarrow{o_i} \sigma_5 \) and \( \gamma' = \sigma_2 \xrightarrow{o_i} \sigma_6 \) in \( P_i \cup \text{Unf}_P(\text{cl}^l) \) such that \( \sigma_2 \) and \( \sigma_6 \) are built-in free configurations, \( \sigma_5 \simeq \sigma'_3 \), \( \sigma_6 \simeq \sigma'_4 \) and such that if \( \gamma \) and \( \gamma' \) use the clause \( cl^l \), then no atoms introduced (in the obvious sense) by \( cl_l \) is rewritten by using (at least) one rule in \( P_i \cup \text{Unf}_P(\text{cl}^l) \). Moreover, by hypothesis and by Lemma 2, \( \sigma_5 \equiv_{Fv(\sigma)} \sigma_6 \).

Let \( l \) be the number of the \textbf{Apply'\textsuperscript{\}} transition steps in \( \delta \) and \( \delta' \), which use the rule \( cl^l \) and whose body is not rewritten by using (at least) one rule in \( P_i \cup \text{Unf}_P(\text{cl}^l) \). The proof is by induction on \( l \).

\( (l = 0) \) In this case, \( \gamma = \sigma_1 \xrightarrow{o_i} \sigma_5 \) and \( \gamma' = \sigma_2 \xrightarrow{o_i} \sigma_6 \) 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 \textbf{Apply'\textsuperscript{\}} transition step in \( \gamma \) and \( \gamma' \), which use (a renamed version of) the rule \( cl^l = r_i \{ H_1 \setminus H_2 \leftrightarrow D \mid K, C ; T \} \), whose body is not rewritten by using (at least) one rule in \( P_i \cup \text{Unf}_P(\text{cl}^l) \) and where \( C \) is the conjunction of all the built-in constraints in the body of \( cl^l \). Without loss of generality, we can assume that such an \textbf{Apply'\textsuperscript{\}} transition step is in \( \gamma \).

Now, we have two possibilities:

- \( \sigma_5 \) is a failed configuration. By definition of \( \equiv_{Fv(\sigma)} \), we have that \( \sigma \) is also a failed configuration. In this case, it is easy to check that, by using Lemma 7, we can substitute each \textbf{Apply'\textsuperscript{\}} transition steps in \( \delta \) and \( \delta' \), which use the rule \( cl^l \) and whose body is not rewritten by using (at least) one rule \( P_i \), with an \textbf{Apply'\textsuperscript{\}} transition step which uses a rule in \( \text{Unf}_P(\text{cl}^l) \subseteq P_{i+1} \). Then, analogous to the case \( (l = 0) \), it is easy to check that there exist the derivations \( \gamma_1 = \sigma_1 \xrightarrow{o_i} \sigma'_5 \).
and \( \gamma'_1 = \sigma_2 \rightarrow \sigma_6 \) in \( P_{t+1} \) such that \( \sigma_3^f \) and \( \sigma_4^f \) are both failed configurations and then we have a contradiction.

\[ \sigma_5 \] is not a failed configuration. Then \( \sigma_5 \) is of the form \( \langle S_5, C_5, T_5 \rangle \), where \( chr(K) \subseteq chr(S_5) \). Moreover, since \( cl^l \) can be non-recursively safely replaced in \( P_t \), there exists a clause \( cl_v \) in \( P_t \setminus \{ cl^l \} \) such that \( cl_v \) can be unfolded by using \( cl_v \). Therefore, by definition of non-recursive safe unfolding, there exists a new derivation \( \gamma_1 = \sigma_1 \rightarrow \sigma_5 \rightarrow \sigma_z' \) such that \( \sigma_z' \) is obtained from \( \sigma_5 \) first by an Apply’ transition step, which uses the rule \( cl_v \) and rewrites atoms in the body of \( cl_v \), and then by some Solve’ transition steps. By the definition of \( \equiv_{Fe(\sigma)} \) and since \( \sigma_5 \equiv_{Fe(\sigma)} \sigma_6 \), we have that there also exists a new derivation \( \gamma'_1 = \sigma_2 \rightarrow \sigma_6 \rightarrow \sigma_6', \) where \( \sigma_6' \) is obtained from \( \sigma_5 \) first by an Apply’ transition step, which uses the rule \( cl_v \) and rewrites atoms in the body of \( cl_v \), and then by some Solve’ transition steps.

Since by hypothesis \( \sigma_5 \equiv_{Fe(\sigma)} \sigma_6 \), we have that \( \sigma_5 \equiv_{Fe(\sigma)} \sigma_6' \). Moreover, the number of the Apply’ transition steps in \( \delta_2 \) and \( \delta_2' \), which use the rule \( cl_v \) whose body is not rewritten by using (at least) one rule in \( P_t \), is strictly less than \( l \) and then the thesis.

- Assume that \( P_{t+1} \) is confluent and let us assume by contrary that \( P_t \) does not satisfy confluence. The proof is analogous to the previous case and hence it is omitted.

\[ \square \]

**Appendix 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 \( \sigma \rightarrow^* \sigma_1^f \not\rightarrow^* \) and \( \sigma \rightarrow^* \sigma_2^f \not\rightarrow^* \), we have \( \sigma_1^f \equiv_{Fe(\sigma)} \sigma_2^f \).

**Proof**

**Only if** The proof is by contradiction. Assume that \( P \) is not confluent. Then there exists a state \( \sigma \) such that \( \sigma \rightarrow^* \sigma_1^f \not\rightarrow^* \) and \( \sigma \rightarrow^* \sigma_2^f \not\rightarrow^* \) and for each pair of states \( \sigma'_1^f \) and \( \sigma''_2^f \) such that \( \sigma_1^f \not\rightarrow^* \sigma'_1^f \) and \( \sigma_2^f \not\rightarrow^* \sigma''_2^f \), we have that \( \sigma'_1^f \not\equiv_{Fe(\sigma)} \sigma''_2^f \). In particular, since \( P \) is normally terminating, we have that there exists \( \sigma'_1^f \) and \( \sigma''_2^f \) such that \( \sigma_1^f \not\rightarrow^* \sigma'_1^f \not\rightarrow^* \sigma''_2^f \not\rightarrow^* \) and \( \sigma'_1^f \not\equiv_{Fe(\sigma)} \sigma''_2^f \). Then it is easy to check that there exist two normal derivations \( \sigma \rightarrow^* \sigma'_1 \not\rightarrow^* \) and \( \sigma \rightarrow^* \sigma''_2 \not\rightarrow^* \) such that \( \sigma'_1 \simeq \sigma'_1 \) and \( \sigma''_2 \simeq \sigma''_2 \). Since \( \sigma'_1 \not\equiv_{Fe(\sigma)} \sigma''_2 \), by definition of \( \simeq \), we have that \( \sigma'_1 \not\equiv_{Fe(\sigma)} \sigma''_2 \) and then we have a contradiction. \[ \square \]

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, 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 $\sigma$ and two normal derivations

$$\sigma \xrightarrow{*_{\omega_i}} \sigma_f \xrightarrow{\omega_i} \sigma \xrightarrow{*_{\omega_i}} \sigma'' \xrightarrow{\omega_i}$$

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

$$\sigma \xrightarrow{*_{\omega_i}} \sigma'_1 \xrightarrow{\omega_i} \sigma' \xrightarrow{*_{\omega_i}} \sigma''_1 \xrightarrow{\omega_i}$$

in $P$, where $\sigma'_1 \simeq \sigma''_1$ and $\sigma''_1 \simeq \sigma''_2$. Since by hypothesis $P$ is confluent, we have that $\sigma'_1 =_{FV(\sigma)} \sigma''_2$. Therefore, by Lemma 2 we have a contradiction to the assumption that there exist two states $\sigma'_1$ and $\sigma''_1$ 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 $r \odot H_1 \setminus H_2 \Leftrightarrow D | A; T$. By Lemma 3 and since by the previous result $P'$ is normally terminating, there exist a state $\sigma$ and two normal derivations

$$\sigma \xrightarrow{*_{\omega_i}} \sigma'_1 \xrightarrow{\omega_i} \sigma \xrightarrow{*_{\omega_i}} \sigma''_1 \xrightarrow{\omega_i}$$

in $P'$ such that $\sigma'_1 \not=_{FV(\sigma)} \sigma''_2$. 

Since \( P' \subseteq P'' \), we have that there exist two normal derivations
\[
\sigma \xrightarrow{\omega_1} \sigma_1^f \quad \text{and} \quad \sigma' \xrightarrow{\omega_2} \sigma_2^f
\]
in \( P'' \). Then, since \( P'' \) is confluent and \( P'' = P' \cup \{ cl \} \) there exists \( i \in [1, 2] \) such that \( \sigma_i^f \xrightarrow{\omega_i} \sigma' \) 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
\[
r \in \mathcal{H}_1 \setminus \mathcal{H}_2 \iff D' \mid A' \mid T'
\]
with \( \mathcal{G}.\mathcal{T} \models D \leftrightarrow D' \) and by construction \( cl' \in P' \). Therefore \( \sigma_i^f \xrightarrow{\omega_i'} \sigma'' \) in \( P' \) by using the rule \( cl' \), and then we have a contradiction. \( \square \)

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, Germany, 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, Germany, 198–231.

DUCK, G. J., STUCKEY, P. J., GARCÍA 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, Germany, 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, Germany, 133–148.

FRÜHWIRTH, T. 2009. Constraint Handling Rules. Cambridge University Press, Cambridge, UK.

FRÜHWIRTH, T. AND ABDENNADHER, S. 2003. Essentials of Constraint Programming. Springer, Berlin, Germany.

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, Amsterdam, Netherlands, 115–130.

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, NY.

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), F. Kaiser and J. Sneyers, Eds., Report CW555 of Katholieke Universiteit Leuven, Department of Computer Science, Heverlee, Belgium, 34–48.
Unfolding for CHR programs

Sarna-Starosta, B. and Schrijvers, T. 2009. Attributed Data for CHR Indexing. In Proceedings of the 25th International Conference on Logic Programming (ICLP 2009), P. M. Hill and D. S. Warren, Eds., Lecture Notes in Computer Science, vol. 5649. Springer, Berlin, Germany, 357–371.

Schrijvers, T. and Sulzmann, M. 2008. Transactions in constraint handling rules. In Proceedings of the 24th International Conference on Logic Programming (ICLP 2008), M. G. de la Banda and E. Pontelli, Eds., Lecture Notes in Computer Science, vol. 5366. Springer, Berlin, Germany, 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 Proceedings of the 21st International Conference (ICLP 2005), M. Gabbielli and G. Gupta, Eds., Lecture Notes in Computer Science, vol. 3668. Springer, Berlin, Germany, 83–97.

Tacchella, P. 2008. Constraint Handling Rules. Compositional Semantics and Program Transformation, PhD thesis, Università di Bologna, Bologna, Italy.

Tacchella, P., Meo, M. C. and Gabbrielli, M. 2007. Unfolding in CHR. In Proceedings of the 9th ACM SIGPLAN International Symposium on Principles and Practice of Declarative Programming (PPDP ’07), M. Leuschel and A. Podelski, Eds., Heverlee, Belgium, 179–186.

Tamaki, H. and Sato, T. 1984. Unfold/fold transformations of logic programs. In Proceedings of the International Conference on Logic Programming, Sten-Åke Tärnlund, Ed., Uppsala University, Uppsala, Sweden, 127–138.

Ueda, K. and Furukawa, K. 1988. Transformation rules for GHC programs. In Proceedings of the International Conference on Fifth Generation Computer Systems 1988 (FGCS’88). ICOT Press, Tokyo, Japan, 582–591.