On the Expressive Power of Priorities in CHR

Maurizio Gabbrielli
Dipartimento di Scienze dell'Informazione
Università di Bologna
Via Mura Anteo Zamboni 7,
40127 Bologna, Italy
gabbri@cs.unibo.it

Jacopo Mauro
Dipartimento di Scienze dell'Informazione
Università di Bologna
Via Mura Anteo Zamboni 7,
40127 Bologna, Italy
jmauro@cs.unibo.it

Maria Chiara Meo
Dipartimento di Scienze
Università di Chieti-Pescara
Viale Pindaro 42,
65127 Pescara, Italy
cmeo@unich.it

Abstract

Constraint Handling Rules (CHR) is a committed-choice declarative language which has been originally designed for writing constraint solvers and which is nowadays a general purpose language.

Recently the language has been extended by introducing user-definable (static or dynamic) rule priorities. The resulting language, called CHR^{rp}, allows a better control over execution while retaining a declarative and flexible style of programming.

In this paper we study the expressive power of this language from the view point of the concurrency theory. We first show that dynamic priorities do not augment the expressive power by providing an encoding of dynamic priorities into static ones. Then we show that, when considering the theoretical operational semantics, CHR^{rp} is strictly more expressive than CHR. This result is obtained by using a problem similar to the leader-election to show that, under some conditions, there exists no encoding of CHR^{rp} into CHR. We also show, by using a similar technique, that the CHR language with the, so called, refined semantics is more expressive power than CHR with theoretical semantics and we extend some previous results showing that CHR can not be encoded into Prolog.

Categories and Subject Descriptors D.3.1 [*Programming languages*]: Formal Definitions and Theory; D.3.2 [*Programming Languages*]: Language Classifications—Concurrent, distributed, and parallel languages; D.3.2 [*Programming Languages*]: Language Classifications - *Constraint and logic languages*; F.3.2 [*Logic and meanings of programs*]: Semantics of Programming Languages

General Terms Languages, Theory.

Keywords Constraint, expressive power.

1. Introduction

Constraint Handling Rules (CHR) [Frühwirth 1991, Frühwirth 1998b] is a committed-choice declarative language which has been originally designed for writing constraint solvers and which is nowadays a general purpose language. A CHR program consists of a set of multi-headed guarded (simplification and propagation) rules which allow one to rewrite constraints into simpler ones until a solved form is reached. The language is parametric w.r.t. an underlying constraint theory $\mathcal{CT}$ which defines the meaning of basic built-in constraints.

The original theoretical operational semantics for CHR (ω_t) is non deterministic, as usual for many other rule based and concurrent languages. Such a non determinism has to be resolved in the implementations by choosing a suitable execution strategy. Most implementations indeed use the, so called, refined operational semantics (ω_r) which has been formalized in [Duck et al. 2004] and fixes most of the execution strategy. This semantics, differently from the theoretical one, offers a good control over execution, however it is quite low-level and lacks flexibility.

For this reason in [De Koninck et al. 2007] an extension of CHR, called CHR^{rp}, was proposed for supporting a high-level, explicit form of execution control which is more flexible and declarative than the one offered by the ω_r semantics. This is obtained by introducing explicitly in the syntax of the language rule annotations, which allow to specify the (static or dynamic) priority of each rule. The operational semantics (called ω_p) is changed accordingly: rules with higher priority are chosen first. Priorities can be either static (the annotations are completely defined at compile time) or dynamic (the annotations contain variables which are instantiated at run-time). Even though in [Sneyers et al. 2009] it is shown that any algorithm can be implemented in CHR preserving time and space complexity, yet in [De Koninck et al. 2007] it is claimed that "priorities do improve the expressivity of CHR".

We provide a formal ground for this informal claim by using a notion of expressivity coming from the field of concurrency theory. In fact, in this field the issue of the expressive power of a language has received a considerable attention in the last years and several techniques and formalisms have been proposed for separating different languages which are Turing powerful (and therefore can not be properly compared by using the standard tools of computability theory). One of these techniques, that we use in this paper, is based on the notion of language encoding [de Boer and Palamidessi 1994, Shapiro 1989, Vaandrager 1993][1] and can be described as follows. Intuitively, a language $\mathcal{L}$ is more expressive than a language $\mathcal{L}'$ or, equivalently, $\mathcal{L}'$ can be encoded in $\mathcal{L}$, if each program written in $\mathcal{L}'$ can be translated into an $\mathcal{L}$ program in such a way that: (1) the intended observable behavior of the original program is preserved (under some suitable decoding); (2) the translation process satisfies some additional restrictions which indicate how easy this process is and how reasonable the decoding of the observables is. For example, typically one requires that the translation is compositional

[1] The original terminology of these papers was "language embedding".

w.r.t. (some of) the syntactic operators of the language [de Boer and Palamidessi 1994].

In this paper we first show that dynamic priorities do no augment the expressive power of CHR^{rp}. This result is shown by providing an encoding of CHR^{rp} programs with dynamic priorities into programs which use only static priorities.

Hence in the following we consider only static priorities and we prove that, when considering the theoretical semantics, CHR^{rp} can not be encoded into CHR under the following two assumptions. First we assume that the observable properties to be preserved are the constraints computed by a program for a goal (more precisely the data sufficient answers). Since these are the typical CHR observables for many CHR reference semantics, assuming their preservation is rather natural. Secondly we require that the translation of a goal is compositional w.r.t. conjunction of goals, that is, we assume that $[\![A, B]\!]_g = [\![A]\!]_g, [\![B]\!]_g$ for any conjunctive goal A, B, where $[\![\ ]\!]_g$ denotes the translation of a goal. We believe this notion of compositionality to be reasonable as well, since essentially it means that the translated program is not specifically designed for a single goal. It is worth noticing that we do not impose any restriction on the translation of the program rules.

In a similar way we also show that the, so called, refined semantics augments the expressive power of CHR w.r.t. the theoretical semantics. This is somehow expected, as this semantics allows additional control over execution and, as shown in [De Koninck et al. 2007] it allows to simulate static priorities (however this simulation does not imply directly our result, as we explain later).

Finally we prove that CHR can not be encoded into Prolog while preserving data sufficient answers, thus extending a previous result provided in [Di Giusto et al. 2009] where only pure Prolog (without built-ins) was considered and where the naive semantics (without token store) rather than the theoretical one was considered.

The remainder of the paper is organized as follows. Next section introduces the languages under consideration with the related semantics and some preliminary notions on language encodings. In Section 3 we provide the translation of dynamic priorities into static ones. In Section 4 we prove that static priorities augment the expressive power of CHR and we also show that the refined semantics augments the expressive power of the language. Section 5 contains the separation result for CHR and Prolog, while Section 6 concludes by discussing some related works.

2. Syntax and semantics

In this section we give an overview of CHR and CHR^{rp} syntax with their operational semantics following [Frühwirth 1998a, Duck et al. 2004, De Koninck et al. 2007].

2.1 Syntax of CHR

A constraint $c(t_1, \ldots, t_n)$ is an atom predicate c/n with t_i a host language value (e.g. a Herbrand term in Prolog) for $1 \leq i \leq n$. There are two types of constraints: built-in (or predefined) constraints that are handled by an existing solver and CHR constraints (or user-defined). Therefore we assume that the signature contains two disjoint sets of predicate symbols for built-in and CHR constraints. For built-in constraints we assume that a (first order) theory $\mathcal{CT}$ describes their meaning.

The notation $\exists_V \phi$, where V is a set of variables, denotes the existential closure of a formula ϕ w.r.t. the variables in V, while the notation $\exists_{-V} \phi$ denotes the existential closure of a formula ϕ with the exception of the variables in V which remain unquantified. $Fv(\phi)$ denotes the free variables appearing in ϕ.

We use $[H \mid T]$ to denote the first H and remaining elements T of a sequence, $++$ for sequence concatenation, $\uplus$ for multi-set union.

reflexivity	$\texttt{leq}(X, Y) \Longleftrightarrow X = Y \mid true$
antisymmetry	$\texttt{leq}(X, Y), \texttt{leq}(Y, X) \Longleftrightarrow X = Y$
transitivity	$\texttt{leq}(X, Y), \texttt{leq}(Y, Z) \Rightarrow \texttt{leq}(X, Z)$

Figure 1. A program for defining $\leq$ in CHR

We follow the logic programming tradition and indicate the application of a substitution σ to a syntactic object t by σt.

To distinguish between different occurrences of syntactically equal constraints, CHR constraints are extended with a unique identifier. An identified CHR constraint is denoted by $c\#i$ with c a CHR constraint and i the identifier. We write $\texttt{chr}(c\#i) = c$ and $\texttt{id}(c\#i) = i$, possibly extended to sets and sequences of identified CHR constraints (or tokens) in the obvious way.

2.2 CHR program

A CHR program is defined as a sequence of three kinds of rules: simplification, propagation and simpagation rules. Intuitively, simplification rewrites constraints into simpler ones, propagation adds new constraints which are logically redundant but may trigger further simplifications, simpagation combines in one rule the effects of both propagation and simplification rules. For simplicity we consider simplification and propagation rules as special cases of a simpagation rule. The general form of a simpagation rule is:

$$r \ @H^k \backslash H^h \Longleftrightarrow g \mid B$$

where r is a unique identifier of a rule, H^k and H^h (the heads) are multi-sets of CHR constraints, g (the guard) is a possibly empty multi-set of built-in constraints and B is a possibly empty multi-set of (built-in and user-defined) constraints. If H^k is empty then the rule is a simplification rule. If H^h is empty then the rule is a propagation rule. At least one of H^k and H^h must be non empty.

In the following when the guard g is empty or $true$ we omit $g \,|$. Also the names of rules are omitted when not needed. For a simplification rule we omit $H^k \backslash$ while we will write a propagation rule as $H^k \Rightarrow g \mid B$. A CHR *goal* is a multi-set of (both user-defined and built-in) constraints. An example of a CHR program is shown in Figure 1.

2.3 Traditional operational semantics

The theoretical operational semantics of CHR, denoted ω_t is given in [Duck et al. 2004] as a state transition system $T = (Conf, \overset{\omega_t}{\rightarrow}_P)$ where configurations in $Conf$ are tuples of the form $\langle G, S, B, T \rangle_n$, where G is the goal (a multi-set of constraints that remain to be solved), S is the CHR store (a set of identified CHR constraints), B is the built-in store (a conjunction of built-in constraints), T is the propagation history (a sequence of identifiers used to store the rule instances fired) and n is the next free identifier (it is used to identify new CHR constraints). The transitions of ω_t are shown in Table 1.

Given a program P, the transition relation $\overset{\omega_t}{\rightarrow}_P \subseteq Conf \times Conf$ is the least relation satisfying the rules in Table 1. The **Solve** transition allows to update the constraint store by taking into account a built-in constraint contained in the goal. The **Introduce** transition is used to move a user-defined constraint from the goal to the CHR constraint store, where it can be handled by applying CHR rules. The **Apply** transition allows to rewrite user-defined constraints (which are in the CHR constraint store) by using rules from the program. As usual, in order to avoid variable name clashes, this transition assumes that all variables appearing in a program clause are fresh ones. The **Apply** transition is applicable when the current store (B) is strong enough to entail the guard of the rule (g), once the parameter passing has been performed.

Solve $\langle \{c\} \uplus G, S, B, T\rangle_n \overset{\omega_t}{\rightarrow}_P \langle G, S, c \wedge B, T\rangle_n$ where c is a built-in constraint

Introduce $\langle \{c\} \uplus G, S, B, T\rangle_n \overset{\omega_t}{\rightarrow}_P \langle G, \{c\#n\} \cup S, B, T\rangle_{n+1}$ where c is a CHR constraint

Apply $\langle G, H_1 \cup H_2 \cup S, B, T\rangle_n \overset{\omega_t}{\rightarrow}_P \langle C \uplus G, H_1 \cup S, \theta \wedge B, T \cup \{t\}\rangle_n$ where P contains a (renamed apart) rule

$$r\ @H_1' \backslash H_2' \Longleftrightarrow g \mid C$$

and there exists a matching substitution θ s.t. $\mathtt{chr}(H_1) = \theta H_1'$, $\mathtt{chr}(H_2) = \theta H_2'$, $\mathcal{CT} \models B \rightarrow \exists_{-Fv(B)}(\theta \wedge g)$ and $t = \mathtt{id}(H_1) +\!\!+ \mathtt{id}(H_2) +\!\!+ [r] \notin T$

Table 1. Transitions of ω_t

An *initial configuration* has the form $\langle G, \emptyset, true, \emptyset\rangle_1$ while a *final configuration* has either the form $\langle G, S, false, T\rangle_k$ when it is *failed*, or the form $\langle \emptyset, S, B, T\rangle_k$ when it is successfully terminated because there are no applicable rules.

Given a goal G, the operational semantics that we consider only observes the non failed final stores of computations terminating with an empty goal and an empty user-defined constraint store. Following the terminology of [Frühwirth 1998a], we call such observables *data sufficient answers*.

Definition 1 (Data sufficient answers)**.** *Let P be a program and let G be a goal. The set $\mathcal{SA}_{P,\omega_t}(G)$ of data sufficient answers for the query G in the program P is defined as:*

$$\mathcal{SA}_{P,\omega_t}(G) \quad = \quad \{\exists_{-Fv(G)} d \,|\mathcal{CT} \not\models d \leftrightarrow false \wedge \wedge \langle G, \emptyset, true, \emptyset\rangle_1 \overset{\omega_t}{\rightarrow}_P^* \langle \emptyset, \emptyset, d, T\rangle_n \overset{\omega_t}{\not\rightarrow}_P\}$$

The previous notion of observables characterizes an input/output behavior, since the input constraint is implicitly considered in the goal.

2.4 Refined operational semantics

The refined operational semantics of CHR, denoted by ω_r is introduced in [Duck et al. 2004] to model the execution mechanism of the current CHR implementations. The refined semantics is based on active constraints. The active constraint is a constraint that is used to find a rule to fire (if any). Only one constraint at the time can be active. The active constraint tries rules in textual order. When a rule instance fires its body is processed from left to right.

We denote the identified constraint that only matches with the j-th occurrence of the constraint c in the program by $c\#i : j$. In the following we will refer to this type of constraints with the name *occurrence identified CHR constraints*. We will define as *execution stack* a sequence of constraints, identified constraints and occurrence identified CHR constraints.

The ω_r semantics of CHR is given as a state transition system $T = (Conf', \overset{\omega_r}{\rightarrow}_P)$ where configurations in $Conf'$ are tuples of the form $\langle A, S, B, T\rangle_n$, where A is an execution stack and S, B, T, n are like in the configurations $Conf$ defined for the traditional semantics.

Given a program P, the transition relation $\overset{\omega_r}{\rightarrow}_P \subseteq Conf' \times Conf'$ is the least relation satisfying the rules in Table 2.

Like in the traditional semantics an *initial configuration* has the form $\langle G, \emptyset, true, \emptyset\rangle_1$ while a *final configuration* has either the form $\langle G, S, false, T\rangle_k$ when it is *failed*, or the form $\langle [\,], S, B, T\rangle_k$ when it is successfully terminated because there are no applicable rules. The difference between these configurations is that while in the traditional semantics the goal is a multi-set of constraints, here the goal is a sequence of constraints. Therefore while in the traditional semantics there is no order between the goal constraints, in the refined semantics the order of the goal constraints is relevant.

Solve $\langle [c|A], S_0 \cup S_1, B, T\rangle_n \overset{\omega_r}{\rightarrow}_P \langle S_1 +\!\!+ A, S_0 \cup S_1, c \wedge B, T\rangle_n$ where c is a built-in constraint and $\mathtt{Fv}(S_0) \subseteq \mathtt{fixed}(B)$ where $\mathtt{fixed}(B)$ are the variables fixed by B

Activate $\langle [c|A], S, B, T\rangle_n \overset{\omega_r}{\rightarrow}_P \langle [c\#n : 1|A], \{c\#n\} \cup S, B, T\rangle_{n+1}$ where c is a CHR constraint

Reactivate $\langle [c\#i|A], S, B, T\rangle_n \overset{\omega_r}{\rightarrow}_P \langle [c\#i : 1|A], S, B, T\rangle_n$ where c is a CHR constraint

Drop $\langle [c\#i : j|A], S, B, T\rangle_n \overset{\omega_r}{\rightarrow}_P \langle A, S, B, T\rangle_n$ where $c\#i : j$ is an occurence identified constraint and there is no occurrence j in P

Simplify $\langle [c\#i : j|A], \{c\#i\} \cup H_1 \cup H_2 \cup H_3 \cup S, B, T\rangle_n \overset{\omega_r}{\rightarrow}_P \langle C +\!\!+ A, H_1 \cup S, \theta \wedge B, T\rangle_n$ where the j^{th} occurrence of a CHR predicate of c in a (renamed apart) rule in P is

$$r\ @H_1' \backslash H_2', d_j, H_3' \Longleftrightarrow g \mid C$$

and there exists a matching substitution θ s.t. $c = \theta d_j$, $\mathtt{chr}(H_1) = \theta H_1'$, $\mathtt{chr}(H_2) = \theta H_2'$, $\mathtt{chr}(H_3) = \theta H_3'$, $\mathcal{CT} \models B \rightarrow \exists_{-Fv(B)}(\theta \wedge g)$

Propagate $\langle [c\#i : j|A], \{c\#i\} \cup H_1 \cup H_2 \cup H_3 \cup S, B, T\rangle_n \overset{\omega_r}{\rightarrow}_P \langle C +\!\!+ [c\#i : j|A], \{c\#i\} \cup H_1 \cup H_2 \cup S, \theta \wedge B, T \cup \{t\}\rangle_n$ where the j^{th} occurrence of a CHR predicate of c in a (renamed apart) rule in P is

$$r\ @H_1', d_j, H_2' \backslash H_3' \Longleftrightarrow g \mid C$$

and there exists a matching substitution θ s.t. $c = \theta d_j$, $\mathtt{chr}(H_1) = \theta H_1'$, $\mathtt{chr}(H_2) = \theta H_2'$, $\mathtt{chr}(H_3) = \theta H_3'$, $\mathcal{CT} \models B \rightarrow \exists_{-Fv(B)}(\theta \wedge g)$ and $t = \langle \mathtt{id}(H_1) +\!\!+ [i] +\!\!+ \mathtt{id}(H_2) +\!\!+ \mathtt{id}(H_3) +\!\!+ [r]\rangle \notin T$

Default $\langle [c\#i : j|A], S, B, T\rangle_n \overset{\omega_r}{\rightarrow}_P \langle [c\#i : j+1|A], S, B, T\rangle_n$ where $c\#i : j$ if no other transition applies

Table 2. Transitions of ω_r

Given a goal G, the operational semantics that we consider observes the non failed final stores of computations terminating with an empty goal and an empty user-defined constraint.

Definition 2 (Data sufficient answers)**.** *Let P be a CHR program and let G be a goal. The set $\mathcal{SA}_{P,\omega_r}(G)$ of data sufficient answers for the query G in the program P is defined as:*

$$\mathcal{SA}_{P,\omega_r}(G) \quad = \quad \{\exists_{-Fv(G)} d \,|\mathcal{CT} \not\models d \leftrightarrow false \wedge \wedge \langle G, \emptyset, true, \emptyset\rangle_1 \overset{\omega_r}{\rightarrow}_P^* \langle [\,], \emptyset, d, T\rangle_n \overset{\omega_r}{\not\rightarrow}_P\}$$

2.5 CHR with priorities

CHR^{rp} is a language introduced in [De Koninck et al. 2007] that extends CHR with user-defined priorities. CHR^{rp} forms an high level alternative for execution control that better suits the need of CHR programmers.

The syntax of CHR^{rp} is compatible with the syntax of CHR. A simpagation rule in CHR^{rp} has the form

$$p :: r\ @H^k \backslash H^h \Longleftrightarrow g \mid B$$

where r, H^k, H^h, g, B are defined as in the CHR simpagation rule in Section 2.2 and p is an arithmetic expression s.t. $\mathtt{Fv}(p) \subseteq (\mathtt{Fv}(H^k) \cup \mathtt{Fv}(H^h))$. A rule where $\mathtt{Fv}(p) = \emptyset$ is called static priority rule, otherwise it is called dynamic. The priority of a static rule is known at compile time while the priority of a dynamic rule is only known at run time.

The formal semantics for CHR^{rp} defined in [De Koninck et al. 2007] is a refinement of the traditional semantics adapted to deal with rule priorities. Formally the semantics of CHR^{rp}, denoted by

Solve $\langle\{c\} \uplus G, S, B, T\rangle_n \overset{\omega_p}{\rightarrow}_P \langle G, S, c \wedge B, T\rangle_n$ where c is a built-in constraint

Introduce $\langle\{c\} \uplus G, S, B, T\rangle_n \overset{\omega_p}{\rightarrow}_P \langle G, \{c\#n\} \cup S, B, T\rangle_{n+1}$ where c is a CHR constraint

Apply $\langle\emptyset, H_1 \cup H_2 \cup S, B, T\rangle_n \overset{\omega_p}{\rightarrow}_P \langle C, H_1 \cup S, \theta \wedge B, T \cup \{t\}\rangle_n$ where P contains a (renamed apart) rule

$$p :: r\ @H_1' \backslash H_2' \Longleftrightarrow g \mid C$$

and there exists a matching substitution θ s.t. $\mathtt{chr}(H_1) = \theta H_1'$, $\mathtt{chr}(H_2) = \theta H_2'$, $\mathcal{CT} \models B \rightarrow \exists_{-Fv(B)}(\theta \wedge g)$ and $t = \mathtt{id}(H_1) +\!\!+ \mathtt{id}(H_2) +\!\!+ [r] \notin T$. Furthermore no rule of priority p' and substitution θ' exists with $\theta' p' < \theta p$ for which the above conditions hold

Table 3. Transitions of ω_p

ω_p, is given as state transition system $T = (Conf, \overset{\omega_p}{\rightarrow}_P)$ where configurations $Conf$, as well as the initial and final configurations are the same as those introduced for the traditional semantics in Section 2.3.

Given a CHR^{rp} program P, the transition relation $\overset{\omega_p}{\rightarrow}_P \subseteq Conf \times Conf$ is the least relation satisfying the rules in Table 3. The **Solve** and **Introduce** transitions are equal to the ones defined for the traditional semantics while the **Apply** transitions is modified by adding a condition that imposes that a rule can be fired if no other rule that can be applied has a smaller value for the priority annotation (as usual in many systems, smaller values correspond to higher priority: for simplicity in the following we will use the terminology "higher" or "lower" priority rather than considering the values).

The data sufficient answers for CHR with priorities can be defined analogously to those of the standard language:

Definition 3 (Data sufficient answers)**.** *Let P be a CHR^{rp} program and let G be a goal. The set $\mathcal{SA}_{P,\omega_p}(G)$ of data sufficient answers for the query G in the program P is defined as:*

$$\begin{aligned}\mathcal{SA}_{P,\omega_p}(G) \quad = \quad & \{\exists_{-Fv(G)} d \,|\, \mathcal{CT} \not\models d \leftrightarrow false \wedge \\ & \wedge \langle G, \emptyset, true, \emptyset\rangle_1 \overset{\omega_p}{\rightarrow}_P^{\,*} \langle\emptyset, \emptyset, d, T\rangle_n \overset{\omega_p}{\not\rightarrow}_P\}\end{aligned}$$

2.6 Language encoding

Since all the variants of CHR that we consider here are Turing powerful [Sneyers et al. 2009], in principle one can always encode a language into another one. The question is how difficult and how acceptable such an encoding is, and depending on the answer to this question one can discriminate different languages. Indeed, CHR^{rp} has been proposed to improve the expressive power of CHR by supporting a high-level form of execution control needed for implementing highly efficient constraint programming systems [De Koninck et al. 2007].

In the fields of distributed algorithms and concurrency the expressiveness issue has received a considerable attention and there exist several techniques that can be used for formally separating different (Turing powerful) languages. The language encoding technique has been discussed in the introduction. Another, related, technique consists in showing that a problem can be solved in a language and not in another. For example, various models of computation have been compared by using the symmetric leader election problem, which consists in requiring the members of a symmetric network to elect one of them as their leader (see [Vigliotti et al. 2007] for a survey).

In all these approaches usually one imposes specific (and hopefully natural) conditions on the encoding which guarantee its meaningfulness. For example, in order to solve the leader election problem the main difficulty is in breaking the initial symmetry, and one does not want that this is achieved by the encoding function. Hence one often requires that the encoding is compositional w.r.t. the operators of the language. Moreover, of course one usually wants that some observable properties of the computations are preserved by the translation.

In the following we will then make similar assumptions on our encoding functions for CHR languages. We formally define a *program encoding* as any function $[\![\ ]\!] : \mathcal{P}_{\mathcal{L}} \rightarrow \mathcal{P}_{\mathcal{L}'}$ which translates an $\mathcal{L}$ program into a (finite) $\mathcal{L}'$ program ($\mathcal{P}_{\mathcal{L}}$ and $\mathcal{P}_{\mathcal{L}'}$ denote the set of $\mathcal{L}$ and $\mathcal{L}'$ programs, respectively). We do not impose any restriction on the program translation.

Next we have to define how the initial goal of the source program has to be translated into the target language. Here we require that the translation is compositional w.r.t. the conjunction of atoms. This assumption essentially means that our encoding respects the structure of the original goal and does not introduce new relations among the variables which appear in the goal. We also require that data sufficient answers are preserved.

Hence we have the following definition where we denote by $\mathcal{G}_{\mathcal{L}}$ and $\mathcal{G}_{\mathcal{L}'}$ the class of $\mathcal{L}$ and $\mathcal{L}'$ goals, respectively (we differentiate these two classes because, for example, a $\mathcal{L}'$ goal could use some user defined predicates which are not allowed in the goals of the original program[2]). Note also that the following definition is parametric w.r.t. a class $\mathcal{G}$ of goals: clearly considering different classes of goals could affect our encodability results. Such a parameter will be instantiated when the notion of acceptable encoding will be used.

Definition 4 (Acceptable encoding)**.** *Let $\mathcal{G} \subseteq \mathcal{G}_{\mathcal{L}}$ be a class of $\mathcal{L}$ goals. An* acceptable encoding *(of $\mathcal{L}$ into $\mathcal{L}'$, for the class of goals $\mathcal{G}$) is a pair of mappings $[\![\ ]\!] : \mathcal{P}_{\mathcal{L}} \rightarrow \mathcal{P}_{\mathcal{L}'}$ and $[\![\ ]\!]_g : \mathcal{G}_{\mathcal{L}} \rightarrow \mathcal{G}_{\mathcal{L}'}$ which satisfy the following conditions:*

- *for any goal $(A, B) \in \mathcal{G}$, $[\![A, B]\!]_g = [\![A]\!]_g, [\![B]\!]_g$ holds.*
- *Data sufficient answers are preserved for the class of goals $\mathcal{G}$, that is, for all $G \in \mathcal{G}$, $\mathcal{SA}_P(G) = \mathcal{SA}_{[\![P]\!]}([\![G]\!]_g)$ holds*[3].

Note that, by a slight abuse of notation, with the first condition in the above definition we actually indicate two cases: when goals are considered as multisets, as in the ω_p semantics and when goals are sequences, as in the refined semantics. In both cases, we only require that the translation of a non atomic goal A, B is the conjunction of the translation of A and of B.

Further weakening this condition and requiring that the translation of A, B is some form of composition of the translation of A and of B does not seem reasonable, as conjunction is the only form for goal composition available in CHR with theoretical semantics, which is the considered target language.

Note that, as previously mentioned, we do not impose any restriction on the program translation (which, in particular, could also be non compositional).

3. Dynamic vs Static Priorities

In this section we prove that the CHR^{rp} language which allows dynamic priorities is not more expressive than the language with static priorities only. This result is obtained by providing an (acceptable)

[2] This means that in principle the signatures of (languages of) the original and the translated program are different.

[3] By a slight abuse of notation here we write $\mathcal{SA}_P$ without indicating the parameter ω_i which refers to the semantics used. Such a parameter will be introduced, in the obvious way, when instantiating this definition with specific cases.

encoding of CHR^{rp} programs with dynamic priorities into CHR^{rp} programs which use only static priorities.

Before showing the encoding $\mathcal{T}(P)$ we need some preliminary notations. We consider a CHR^{rp} program P composed by m rules and we assume that the i-th rule (with $i \in \{1,\dots,m\}$) has the form:

$$p_i :: rule_i \ @ \ H_i \backslash H'_i \Leftrightarrow G_i | C_i$$

We suppose that $Head(P)$ are all the names c/n s.t. $c(t_1,\dots,t_n)$ is in the head of a rule in P.

In the following we denote by $\bar{t}$ and $\bar{X}$ a sequence of terms (i.e. host language values) and distinct variables, respectively. Moreover, given $H = c_1(\bar{t}_1),\dots,c_n(\bar{t}_n)$ we use $H(t'_1,\dots,t'_n)$ for the sequence of constraints $c'_1(t'_1,\bar{t}_1),\dots,c'_n(t'_n,\bar{t}_n)$.

Given a CHR^{rp} program P which uses dynamic priorities, its encoding $\mathcal{T}(P)$ into CHR^{rp} with static priorities only is the program consisting of the rules t_i and $t_{i,j}$ defined as follows:

for every predicate name $c/n \in Head(P)$
$$1 :: t_{1,c/n} \ @ \ start \backslash state\_id(Z), c(\bar{X}) \Leftrightarrow c'(Z,\bar{X}), state\_id(Z+1)$$

for every predicate name $c/n \in Head(P)$
$$2 :: t_{2,c/n} \ @ \ c(\bar{X}) \Rightarrow start, state\_id(0)$$

$$2 :: t_3 \ @ \ start \Leftrightarrow highest\_priority(inf)$$

for every $i \in \{1,\dots,m\}$
$$3 :: t_{4,i} \ @ \ end \backslash instance_i(\bar{V}) \Leftrightarrow true$$

$$4 :: t_5 \ @ \ end \Leftrightarrow true$$

for every $i \in \{1,\dots,m\}$ `EVALUATE_PRIORITIES`(i)

$$7 :: t_9 \ @ \ highest\_priority(inf), state\_id(Z) \Leftrightarrow end$$

for every $i \in \{1,\dots,m\}$ `ACTIVATE_RULE`(i)

If i is not a propagation rule then `EVALUATE_PRIORITIES`(i) are the following rules

$$6 :: t_{7,i} \ @ \ H_i(\bar{U}), H'_i(\bar{V}) \backslash highest\_priority(inf) \Leftrightarrow G_i | highest\_priority(p_i)$$

$$6 :: t_{8,i} \ @ \ H_i(\bar{U}), H'_i(\bar{V}) \backslash highest\_priority(P) \Leftrightarrow G_i, p_i < P | highest\_priority(p_i)$$

if i is a propagation rule then `EVALUATE_PRIORITIES`(i) are the following rules

$$5 :: t_{6,i} \ @ H_i(\bar{V}) \Rightarrow G_i | instance_i(\bar{V})$$

$$6 :: t_{7,i} \ @ \ instance_i(\bar{V}), H_i(\bar{V}) \backslash highest\_priority(inf) \Leftrightarrow G_i | highest\_priority(p_i)$$

$$6 :: t_{8,i} \ @ \ instance_i(\bar{V}), H_i(\bar{V}) \backslash highest\_priority(P) \Leftrightarrow G_i, p_i < P | highest\_priority(p_i)$$

if i is a propagation rule then `ACTIVATE_RULE`(i) is the following rule

$$8 :: t_{10,i} \ @ \ H_i(\bar{V}) \backslash instance_i(\bar{V}), highest\_priority(P), state\_id(Z) \Leftrightarrow G_i, p_i = P | Update(C_i, Z), highest\_priority(inf)$$

if i is not a propagation rule then `ACTIVATE_RULE`(i) is the following rule

$$8 :: t_{10,i} \ @ \ H_i(\bar{U}) \backslash H'_i(\bar{V}), highest\_priority(P), state\_id(Z) \Leftrightarrow G_i, p_i = P | Update(C_i, Z), highest\_priority(inf)$$

where $Update(C, Z)$ is defined as follows

$$Update(c(\bar{X}), Z) = c'(Z, \bar{X})$$
if c is a CHR constraint

$$Update(b(\bar{X}), Z) = b(\bar{X})$$
if b is a built-in constraint

$$Update([\,], Z) = state\_id(Z)$$

$$Update([d(\bar{X}) \mid Xs], Z) = Update(d(\bar{X}), Z), Update(Xs, Z+1)$$

In the above encoding we assume that the constraint theory $\mathcal{CT}$ allows to use equalities and inequalities (so we can evaluate whether $p_i = h$ and $p_i > h$ where $h \in \mathbb{Z}$ and p_i is an arithmetic expression). We also assume inf is a conventional constant which is bigger than all p_i (i.e. it represents the lowest priority). We also assume that `New_names`(P) =

$$\{highest\_priority, start, end, state\_id\} \cup \cup \{c' \mid c \in Head(P)\} \cup \{instance_i \mid i \in \{1,\dots m\}\}$$

is a set of fresh predicate names, that we use in our encoding, which are not used elsewhere in programs and goals (note that this assumption is a weak restriction, since our programs are finite and we can always chose a set of predicates which do not appear in a program).

We now provide some explanations for the above encoding.

Intuitively the result of the encoding can be divided in three phases: The first one is the *init* phase and is composed by rules $t_{1,c/n} \dots t_3$; the second one, called *main*, is in its turn composed by two phases: *evaluate* containing rules $t_{6,i} \dots t_{8,i}$ and *activation* which contains rules $t_{10,i}$; the third phases is the *termination* one and is composed by rules $t_{4,i}, t_5, t_9$.

We first describe these three phases in general.

Init; In the init phase, for each (user defined) constraint predicate c which appears in the head of a rule in the original program P we introduce a rule $t_{1,c/n}$ which replaces $c(\bar{X})$ (where $\bar{X}$ is a tuple of distinct variables) by $c'(Z,\bar{X})$ where c' is a new predicate in `New_names (P)` and Z is a variable which will be used to simulate the identifier used in identified constraints (to be used in the propagation history of the ω_p semantics). Moreover we use a *state_id* predicate to memorize the highest identifier used. Rules $t_{2,c/n}$ (one for each predicate c, as before) are used to fire rules $t_{1,c/n}$ and also to start the following phase (via rule t_3). Note that rules $t_{1,c/n}$ have maximal priority and therefore are tried before rules $t_{2,c/n}$.

Main; As previously mentioned, the main is divided into two phases: the *evaluation* phase starts when the init phase adds the constraint $highest\_priority(inf)$. In this part rules $t_{6,i} \dots t_{8,i}$ store in $highest\_priority$ the highest priority on all the rule instances that can be fired. After the end of the evaluation phase the *activation* starts. During this phase if a rule can be fired one of the rules $t_{10,i}$ is fired. After the rule has been fired the constraint $highest\_priority(inf)$ is produced which starts a new evaluation phase.

Termination; The termination phase is triggered by rule t_9. This rule fires when no instance from the original program can fire. During the termination phase all the constraints produced during the computation (namely $state\_id$, $instance_i$, $highest\_priority$, end) are deleted.

In the following we now provide some more details on the two crucial points in this translation, namely the evaluation and the activation phases (contained in the Main phase).

Evaluation; The rules in the set denoted by

$$\texttt{EVALUATE\_PRIORITIES}(i)$$

are triggered by the insertion of $highest\_priority(inf)$ in the constraint store. We describe below first the case of propagation rules and then the one of simpagation and simplification ones.

In the case of a propagation rule i, the rules in

$$\texttt{EVALUATE\_PRIORITIES}(i)$$

should consider the possibility that there is an instance of i that can not be fired because it has been previously fired. When an instance of a propagation rule can fire, rule $t_{6,i}$ adds a constraint $instance_i(\bar{v})$, where $\bar{v}$ are the identifiers of the rule instance. The absence of the constraint $instance_i(\bar{v})$ in the constraint store means that the rule i can not be fired or has already fired for the CHR atoms identified by $\bar{v}$.

The evaluation of the priority for a simpagation or a simplification rule is simpler because the propagation history does not affect the execution of these two types of rules.

Rules $t_{7,i}$ and $t_{8,i}$ replace the constraint $highest\_priority(p)$ with the constraint $highest\_priority(p')$ if a rule of priority p' can be fired and $p > p'$.

Activation; When the evaluation phase ends if a rule can fire then one of the rules $t_{10,i}$ is fired since $highest\_priority(inf)$ can not be in the store.

The only difference between a propagation rule and a simpagation/simplification rule is that when a propagation rule is fired the corresponding constraint $instance_i(\bar{V})$ is deleted to avoid the execution of the same propagation rule in the future.

It is worth noting that the non-determinism in the choice of the rule to be fired provided by the semantics ω_p is preserved, since all the priorities of $\texttt{ACTIVATE\_RULE}(i)$ are equal.

The following result shows that the data sufficient answers are preserved by our encoding. Its proof follows the lines of the reasoning informally explained above.

Theorem 1. *Assume that the predicate symbols in* $\texttt{New\_name}(P)$ *are not used in a CHR*^{rp} *program P and in a goal G. Then* $\mathcal{SA}_{P,\omega_p}(G) = \mathcal{SA}_{\mathcal{T}(P),\omega_p}(G)$ *holds.*

Since the translation of the goal is compositional (it is the identity function) and the translated program does not contain dynamic priorities, we have the following immediate corollary.

Corollary 1. *Assume that the predicate symbols in* $\texttt{New\_name}(P)$ *are not used in CHR*^{rp} *programs with dynamic priorities and in goals, then $\mathcal{T}(P)$, together with the identity on goals, is an acceptable encoding of CHR*^{rp} *with dynamic priorities into CHR*^{rp} *with static priorities.*

It is worth noting that if the original program is confluent then also its translation $\mathcal{T}(P)$ is confluent.

4. CHR vs CHR^{rp}

In order to prove our first separation result we need the following lemma which states a key property of CHR computations under the ω_t semantics. Essentially it says that if the goal G produces a data sufficient answer d, then if the goal is replicated there exists a computation that will terminate producing the same data sufficient answer d.

Lemma 1. *Let P be a CHR program. Then* $\mathcal{SA}_{P,\omega_t}(G) \subseteq \mathcal{SA}_{P,\omega_t}(G \cup G)$.

Proof. If $\mathcal{SA}_{P,\omega_t}(G) = \emptyset$ then the lemma holds trivially. Conversely if d is in $\mathcal{SA}_{P,\omega_t}(G)$ then there is a derivation

$$\langle G, \emptyset, true, \emptyset\rangle_1 \overset{\omega_t}{\rightarrow}{}^*_P \langle \emptyset, \emptyset, B, T\rangle_k$$

s.t. $\mathcal{CT} \not\models B \leftrightarrow false$ and $d = \exists_{-Fv(G)}B$. Then the lemma holds because starting from a state $\langle G, \emptyset, true, T\rangle_k$ the first apply transition is executable only after at least an introduce transition that increases the state identifier. Since T cannot contain a sequence $[i_1, \ldots, i_l, r]$ s.t. i_j is either greater than or equal to k for all $j \in \{1, \ldots, l\}$ then all the transactions in $\langle G, \emptyset, true, \emptyset\rangle_1 \overset{\omega_t}{\rightarrow}{}^*_P \langle \emptyset, \emptyset, B, T\rangle_k$ can be executed starting from the state $\langle G, \emptyset, B, T\rangle_k$. □

Lemma 1 is not true anymore if we consider CHR^{rp} programs. Indeed if we consider the program P consisting of the rules

$$1 :: a, a \Leftrightarrow false$$

$$2 :: a \Leftrightarrow true$$

then the goal a has the data sufficient answer $true$ in P, while the goal (a, a) has no data sufficient answer in P. With the help of the previous lemma we can now prove our main separation result. This is obtained by showing that, given a goal consisting of n repeated atoms, in CHR^{rp} one can write a program which answers "yes" if $n = 1$ and "no" otherwise. This is similar to the, so called, Last Man Standing problem which has been introduced in [Versari et al. 2007]: a set of n processes composing a network must realize, in a distributed way, if $n = 1$ or $n > 1$. Clearly our setting is different, since while in [Versari et al. 2007] a process algebraic view is considered, hence the problem has to be solved by using explicit communications among processes, here we consider a rule based language where communication among different "processes" (i.e. different atoms in the goal) happens indirectly through a common store. However it is worth noting that, in both cases, the presence of priorities does augment the expressive power because it allows to check for *absence* rather than presence of information. For example, considering CHR with the ω_t semantics, if the guard of a rule r is not satisfied in the present store (and is not inconsistent with it) then the computation for r is blocked and there is no way to specify that some alternative action must be taken (of course one could use another rule r' whose guard is satisfied, but then such a rule can be used also when the guard of r is satisfied). So, one can not express something like "if c is not present then do the following". On the contrary this can be expressed by using priorities and a suitable ordering of clauses, as appears from the program in the proof of the following.

Theorem 2. *Let $\mathcal{G}$ be a class of goals such that if H is a head of a rule then $H \in \mathcal{G}$. There exists no acceptable encoding of CHR*^{rp} *in CHR for the class $\mathcal{G}$.*

Proof. The proof is by contradiction. Consider the following program P in CHR^{rp}

$$1 :: a(X), a(X) \Leftrightarrow X = no$$

$$2 :: a(X) \Leftrightarrow X = no | true$$

$$3 :: a(X) \Leftrightarrow X = yes$$

and assume that $[\![P]\!]$ is the translation of P in CHR.

Let G be a non empty goal of the form $a(X), \ldots, a(X)$. Then $\mathcal{SA}_{P,\omega_p}(G) = \{\{X = yes\}\}$ iff G contains only one occurrence of the atom $a(X)$ and $\mathcal{SA}_{P,\omega_p}(G) = \{\{X = no\}\}$ otherwise.

Since the goal $a(X)$ has the data sufficient answer $\{X = yes\}$ in the program P and since the encoding preserves data sufficient answers, the goal $[\![a(X)]\!]_g$ has the data sufficient answer $\{X = yes\}$ also in the program $[\![P]\!]$. From the compositionality of the translation of goals and previous Lemma 1 it follows that the goal $[\![a(X), a(X)]\!]_g = [\![a(X)]\!]_g, [\![a(X)]\!]_g$ has the data sufficient answer $\{X = yes\}$ in the encoded program $[\![P]\!]$. However $a(X), a(X)$ has only the data sufficient answer $\{X = no\}$ in the original program P. This contradicts the fact that $[\![P]\!]$ is an acceptable encoding for P, thus concluding the proof. □

4.1 Considering the refined semantics

In [De Koninck et al. 2007] it was shown that CHR^{rp} programs with static priorities can be translated into CHR, provided that for the latter language one considers the refined semantics (ω_r) rather than the theoretical one (ω_t). Indeed, as discussed in that paper, the refined semantics provides an additional control over execution (even though at a low level) and it therefore allows to simulate (static) priorities.

Here we prove that indeed the refined semantics does augment the expressive power of CHR w.r.t. the theoretical semantics. In fact we show that there is no acceptable encoding of CHR language with the ω_r semantics (denoted by CHR$_{\omega_r}$ in the following) into CHR with the ω_t semantics (denoted by CHR$_{\omega_t}$). The proof is analogous to that one of the previous theorem. Note however that Theorem 2, together with the translation of CHR^{rp} into CHR$_{\omega_r}$ provided in [De Koninck et al. 2007] does not imply the following result, since that translation does not consider compositionality[4].

Theorem 3. *Let $\mathcal{G}$ be a class of goals such that if H is a head of a rule then $H \in \mathcal{G}$. There exists no acceptable encoding of CHR$_{\omega_r}$ into CHR$_{\omega_t}$ for the class $\mathcal{G}$.*

Proof. The proof is by contradiction. Consider the following program P in CHR$_{\omega_r}$ [5]

$$a(X) \Leftrightarrow X = no | true$$

$$a(X) \Leftrightarrow X = yes | false$$

$$d(X), b(X), a(X) \Leftrightarrow X = no$$

$$a(X) \Leftrightarrow b(Y), b(X), c(X)$$

$$c(X), b(Y) \Leftrightarrow Y = yes, d(X)$$

$$d(X), b(Y) \Leftrightarrow X = yes | true$$

and assume that $[\![P]\!]$ is the translation of P in CHR$_{\omega_t}$.

Let G be a non empty goal of the form $a(X), \ldots, a(X)$. Then $\mathcal{SA}_{P,\omega_r}(G) = \{\{X = yes\}\}$ if G contains only one occurrence of the atom $a(X)$, $\mathcal{SA}_{P,\omega_r}(G) = \{\{X = no\}\}$ otherwise.

Since the goal $a(x)$ has the data sufficient answer $\{X = yes\}$ in the program P and since the encoding preserves data sufficient answers, $\{X = yes\} \in \mathcal{SA}_{[\![P]\!],\omega_t}([\![a(X)]\!]_g)$. As in Theorem 2, from the compositionality of the translation of goals and Lemma 1 it follows that $\{X = yes\} \in \mathcal{SA}_{[\![P]\!],\omega_t}([\![a(X), a(X)]\!]_g)$. This contradicts the fact that $[\![P]\!]$ is an acceptable encoding for P, thus concluding the proof. □

[4] Moreover the correctness result for the translation given by Theorem 4 in [De Koninck et al. 2007] shows only that the derivations in the translated program correspond to derivations in the original program, and not vice versa.

[5] To prove Theorem 3 it is possible to use programs with fewer rules. We chose instead to use a program that solves the Last Men Standing problem like in Theorem 2

5. CHR vs Prolog

We prove now that CHR can not be encoded into Prolog while preserving data sufficient answers. We extend a previous result provided in [Di Giusto et al. 2009] where only pure Prolog (without built-ins) was considered. In fact, such a result was based on a property of computations (Lemma 1 in [Di Giusto et al. 2009]) which does not hold for some Prolog built-ins such as `var`. Moreover, in [Di Giusto et al. 2009] the naive semantics (without token store) was considered, while here we assume that CHR uses the ω_t semantics. Here we provide a direct, more general result which holds for Prolog systems that do not have dynamic procedures (i.e. it is not possible to use built-ins like assert, retract, clause, ...) considering the semantics based on computed answers.

First we have to define more precisely what is an acceptable encoding of CHR$_{\omega_t}$ (CHR with the ω_t semantics) into Prolog. In fact, since constraints are computed in CHR while substitutions are computed in Prolog, one has to define the decoding functions which allow to transform an observable into the other, hence slightly modifying the second condition of Definition 4. Here we impose a very weak requirement on such decodings, namely we assume that they can not transform an empty set of observables into a non empty one. This is a rather reasonable assumption, since clearly a function, which creates an answer from an empty set, appears to be "suspicious". Hence we have the following definition where, for the convenience of the reader, we repeat also the unchanged part of the previous Definition 4. We denote by $\mathcal{CA}_P(G)$ the set of computed answer substitutions for the goal G in the (Prolog) program P.

Definition 5 (Acceptable encoding of CHR into Prolog)**.** *Let $\mathcal{G}$ be a class of CHR$_{\omega_t}$ goals. An* acceptable encoding *of CHR$_{\omega_t}$ into Prolog, for the class of goals $\mathcal{G}$, is a pair of mappings $[\![\ ]\!] : \mathcal{P}_{CHR_{\omega_t}} \to \mathcal{P}_{Prolog}$ and $[\![\ ]\!]_g : \mathcal{G}_{CHR_{\omega_t}} \to \mathcal{G}_{Prolog}$ which satisfy the following conditions:*

- *for any goal $(A, B) \in \mathcal{G}$, $[\![A, B]\!]_g = [\![A]\!]_g, [\![B]\!]_g$,*
- *$\mathcal{CA}_{[\![P]\!]}([\![G]\!]_g) = \emptyset$ if and only if $\mathcal{SA}_{P,\omega_t}(G) = \emptyset$.*

Hence we have the following separation result.

Theorem 4. *Let $\mathcal{G}$ be a class of CHR$_{\omega_t}$ goals such that if H is a head of a rule then $H \in \mathcal{G}$. There exists no acceptable encoding of CHR$_{\omega_t}$ in Prolog for the class $\mathcal{G}$.*

Proof. The proof is by contradiction. Consider the following CHR$_{\omega_t}$ program P:

$$a, b \Leftrightarrow true$$

where a, b are CHR constraints, and assume that $[\![P]\!]$ is the translation of P in Prolog.

We have that $\mathcal{SA}_{P,\omega_t}(a) = \emptyset$ while $\mathcal{SA}_{P,\omega_t}((a, b)) = \{\{true\}\}$. By previous definition of *acceptable encoding* it follows that the goal $[\![a]\!]_g$ in the Prolog program $[\![P]\!]$ has no computed answer substitution, while the goal $[\![a, b]\!]_g$ in $[\![P]\!]$ has at least one computed answer substitution. From the compositionality of $[\![\ ]\!]_g$, we have that $[\![a]\!]_g, [\![b]\!]_g = [\![a, b]\!]_g$ has an answer substitution in $[\![P]\!]$.

Without loss of generality, let us consider the standard leftmost selection rule of Prolog (i.e the atom chosen for the SLD resolution is always the leftmost in the goal). Starting from the goal $[\![a]\!]_g, [\![b]\!]_g$ there exists a successful derivation

$$[\![a]\!]_g, [\![b]\!]_g \to^*_{[\![P]\!]} [\![b']\!]_g \to^*_{[\![P]\!]} \Box$$

where all the sub-goal derived from $[\![a]\!]_g$ have been reduced to the empty goal before considering the (possibly instantiated version of) $[\![b]\!]_g$ (here $\Box$ is the empty goal and $\to_{[\![P]\!]}$ is an SLD resolution step in the program $[\![P]\!]$).

This implies that there exists a successful derivation

$$[\![a]\!]_g \rightarrow^*_{[\![P]\!]} \Box$$

and therefore the goal $[\![a]\!]_g$ in the Prolog program $[\![P]\!]$ has one computed answer substitution. However the goal a has no data sufficient answer in P, thus contradicting the fact that $[\![P]\!]$ was an acceptable encoding for P. $\Box$

6. Conclusions

We have studied the expressive power of CHR with priorities and we have shown that dynamic priorities do not increase the expressive power of static ones. On the other hand we have proved that, when considering the theoretical semantics, CHR with (static) priorities can not be encoded into standard CHR under quite reasonable assumptions. A similar result shows also that the refined semantics allows to augment the expressive power of CHR w.r.t. the theoretical one. Finally we have shown that CHR can not be encoded in Prolog, thus extending a previous result provided in [Di Giusto et al. 2009] where only pure Prolog (without built-ins) was considered. Indeed the present work can be seen as a continuation of that paper where it was also shown that CHR can not be encoded into CHR with single heads.

As mentioned in the introduction, this paper is also based on [De Koninck et al. 2007], where the CHR^{rp} language was introduced and where it was shown that CHR^{rp} programs with static priorities can be translated into CHR with the refined semantics. However that paper did not provide the formal results that we have shown here.

Among the other few papers which consider the expressive power of CHR a quite relevant one is [Sneyers et al. 2009], where the authors show that it is possible to implement any algorithm in CHR in an efficient way, i.e. with the best known time and space complexity. This result is obtained by introducing a new model of computation, called the CHR machine, and comparing it with the well-known Turing machine and RAM machine models. Earlier works by Frühwirth [Frühwirth 2002, 2001] studied the time complexity of simplification rules for naive implementations of CHR. In this approach an upper bound on the derivation length, combined with a worst-case estimate of (the number and cost of) rule application attempts, allows to obtain an upper bound of the time complexity. The aim of all these works is different from ours, even though they can be used to state that, in terms of classical computation theory, CHR^{rp} is equivalent to CHR.

Another recent paper which studies the expressive power of CHR is [Sneyers 2008], where the author shows that several subclasses of CHR are still Turing-complete, while single-headed CHR without host language and propositional abstract CHR are not Turing-complete.

In the field of concurrent languages one can find several works related to the present one. In particular, concerning priorities, a recent paper [Versari et al. 2007] shows that the presence of priorities in process algebras does augment the expressive power. More precisely the authors show, among other things, that a finite fragment of asynchronous CCS with (global) priority can not be encoded into π-calculus nor in the broadcast based b-π calculus. This result is proved by introducing a formalization of the Last Man Standing problem which is somehow similar to the problem that we have used to separate CHR^{rp} and CHR, even though the formal setting is rather different.

More generally, often in process calculi and in distributed systems separation results are obtained by showing that a problem can be solved in a language and not in another one (under some additional hypothesis, similar to those used here). For example, in [Palamidessi 2003] the author proves that there exists no *reasonable* encoding from the π-calculus to the asynchronous π-calculus by showing that the symmetric leader election problem has no solution in the asynchronous version of the π-calculus. A survey on separation results based on this problem can be found [Vigliotti et al. 2007].

We are extending this work along several lines. First we are considering similar results for qualified answers. Moreover we are considering whether some complexity results can be obtained for our translation of dynamic into static priorities.

Finally we plan to investigate the relation between priorities and negation as absence [Van Weert et al. 2006]. In fact, as mentioned in Section 4, by using priorities one can check the absence of information. Threfore it seems that one can encode negation as absence in CHR^{rp}.

References

Frank S. de Boer and Catuscia Palamidessi. Embedding as a tool for language comparison. *Information and Computation*, 108(1):128–157, 1994.

Leslie De Koninck, Tom Schrijvers, and Bart Demoen. User-definable rule priorities for chr. In Michael Leuschel and Andreas Podelski, editors, *PPDP*, pages 25–36. ACM, 2007. ISBN 978-1-59593-769-8.

Cinzia Di Giusto, Maurizio Gabbrielli, and Maria Chiara Meo. Expressiveness of multiple heads in chr. In Mogens Nielsen, Antonín Kucera, Peter Bro Miltersen, Catuscia Palamidessi, Petr Tuma, and Frank D. Valencia, editors, *SOFSEM*, volume 5404 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 2009. ISBN 978-3-540-95890-1.

Gregory J. Duck, Peter J. Stuckey, Maria J. García de la Banda, and Christian Holzbaur. The refined operational semantics of constraint handling rules. In Bart Demoen and Vladimir Lifschitz, editors, *ICLP*, volume 3132 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2004. ISBN 3-540-22671-0.

T. Frühwirth. As time goes by: Automatic complexity analysis of simplification rules. In *KR 02*, 2002.

Thom Frühwirth. Introducing simplification rules. Technical report, 1991.

Thom W. Frühwirth. Theory and practice of constraint handling rules. *J. Log. Program.*, 37(1-3):95–138, 1998a.

Thom W. Frühwirth. As time goes by II: More automatic complexity analysis of concurrent rule programs. *Electr. Notes Theor. Comput. Sci.*, 59(3), 2001.

Thom W. Frühwirth. Theory and practice of constraint handling rules. *J. Log. Program.*, 37(1-3):95–138, 1998b.

Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous *pi*-calculi. *Mathematical. Structures in Comp. Sci.*, 13(5):685–719, 2003. ISSN 0960-1295.

Ehud Y. Shapiro. The family of concurrent logic programming languages. *ACM Comput. Surv.*, 21(3):413–510, 1989.

Jon Sneyers. Turing-complete subclasses of chr. In Maria Garcia de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 759–763. Springer, 2008. ISBN 978-3-540-89981-5.

Jon Sneyers, Tom Schrijvers, and Bart Demoen. The computational power and complexity of constraint handling rules. *ACM Trans. Program. Lang. Syst.*, 31(2), 2009.

Frits W. Vaandrager. Expressive results for process algebras. In *Proceedings of the REX Workshop on Sematics: Foundations and Applications*, pages 609–638, London, UK, 1993. Springer-Verlag. ISBN 3-540-56596-5.

Peter Van Weert, Jon Sneyers, Tom Schrijvers, and Bart Demoen. Extending chr with negation as absence. In *Proceedings of CHR 06*, Venice, Italy, July 2006.

Cristian Versari, Nadia Busi, and Roberto Gorrieri. On the expressive power of global and local priority in process calculi. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2007. ISBN 978-3-540-74406-1.

Maria Grazia Vigliotti, Iain Phillips, and Catuscia Palamidessi. Tutorial on separation results in process calculi via leader election problems. *Theor. Comput. Sci.*, 388(1-3):267–289, 2007. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/j.tcs.2007.09.001.