TLP **10** (4–6): 611–626, 2010. © Cambridge University Press 2010
doi:10.1017/S1471068410000311

Decidability properties for fragments of CHR

MAURIZIO GABBRIELLI
Dipartimento di Scienze dell'Informazione and Lab. Focus INRIA, Università di Bologna, Italy
(*e-mail:* gabbri@cs.unibo.it)

JACOPO MAURO
Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy
(*e-mail:* jmauro@cs.unibo.it)

MARIA CHIARA MEO
Dipartimento di Scienze, Università di Chieti Pescara, Italy
(*e-mail:* cmeo@unich.it)

JON SNEYERS*
Departement Computerwetenschappen, K.U. Leuven, Belgium
(*e-mail:* jon.sneyers@cs.kuleuven.be)

submitted 7 February 2010; revised 10 April 2010; accepted 17 May 2010

Abstract

We study the decidability of termination for two CHR dialects which, similarly to the Datalog like languages, are defined by using a signature which does not allow function symbols (of arity > 0). Both languages allow the use of the $=$ built-in in the body of rules, thus are built on a host language that supports unification. However each imposes one further restriction. The first CHR dialect allows only *range-restricted* rules, that is, it does not allow the use of variables in the body or in the guard of a rule if they do not appear in the head. We show that the existence of an infinite computation is decidable for this dialect. The second dialect instead limits the number of atoms in the head of rules to one. We prove that in this case, the existence of a terminating computation is decidable. These results show that both dialects are strictly less expressive[1] than Turing Machines. It is worth noting that the language (without function symbols) without these restrictions is as expressive as Turing Machines.

KEYWORDS: constraint programming, expressivity, well-structured transition systems

1 Introduction

Constraint Handling Rules (CHR; Frühwirth 1998, 2009) is a declarative general-purpose language. A CHR program consists of a set of multi-headed guarded (simplification, propagation and simpagation) rules which allow one to rewrite constraints into simpler ones until a solved form is reached. The language is

* This research was partially supported by the MIUR PRIN 20089M932N project: "Innovative and multi-disciplinary approaches for constraint and preference reasoning".
[1] As we clarify later, "less expressive" here means that there exists no termination preserving encoding of Turing machines in the considered language.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

parametric w.r.t. an underlying constraint theory $\mathcal{CT}$ which defines basic built-in constraints. For a recent survey on the language see Sneyers *et al.* (2010).

In the last few years, several papers have investigated the expressivity of CHR, however very few decidability results for fragments of CHR have been obtained. Three main aspects affect the computational power of CHR: the number of atoms allowed in the heads, the nature of the underlying signature on which programs are defined, and the constraint theory. The latter two aspects are often referred to as the "host language" since they identify the language on which a CHR system is built. Some results in (Di Giusto *et al.* 2009) indicate that restricting to single-headed rules decreases the computational power of CHR. However, these results consider Turing complete fragments of CHR, hence they do not establish any decidability result. Indeed, single-headed CHR is Turing-complete (Di Giusto *et al.* 2009), provided that the host language allows functors and unification. On the other hand, when allowing multiple heads, even restricting to a host language which allows only constants does not allow to obtain any decidability property, since even with this limitation CHR is Turing complete (Sneyers 2008; Di Giusto *et al.* 2009). The only (implicit) decidability results concern propositional CHR, where all constraints have arity 0, and CHR without functors and without unification, since these languages can be translated to (colored) Petri Nets (Betz 2007)—see also Section 5.

Given this situation, when looking for decidable properties it is natural to consider further restrictions of the above mentioned CHR language which allows the only built-in $=$ (interpreted in the usual way as equality on the Herbrand universe) and which, similarly to Datalog, is defined over a signature which contains no function symbol of arity > 0. We denote such a language by CHR(C).

In this paper we provide two decidability results for two fragments of CHR(C). The first fragment allows *range-restricted* rules only, that is, it does not allow the use of a variable in the body or in the guard if it does not appear in the head. We show, using the theory of well-structured transition systems (Finkel and Schnoebelen 2001; Abdulla *et al.* 1996), that in this case the existence of an infinite computation is decidable. The second fragment that we consider is single-headed CHR(C), denoted by $\text{CHR}_1(C)$. We prove that, for this language, the existence of a terminating computation is decidable. In this case we provide a direct proof, since no reduction to Petri Nets can be used (the language introduces an infinite states system) and well-structured transition system can not be used (they do not allow to prove this kind of decidability properties).

These results show that both CHR fragments are strictly less expressive than Turing Machines. As previously mentioned, CHR(C) is as expressive as Turing Machines. So these results obviously imply that both restrictions lower the expressive power of CHR(C).

2 Syntax and semantics

In this section we give an overview of CHR syntax and its operational semantics following (Frühwirth 1998; Duck *et al.* 2004). A constraint $c(t_1, \ldots, t_n)$ is an atomic formula constructed on a given signature Σ in the usual way. There are two types of constraints: built-in constraints (predefined) that are handled by an existing solver

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

and CHR constraints (user-defined) which are defined by a CHR program. Therefore we assume that the signature Σ contains two disjoint sets of predicate symbols for built-in and CHR constraints. For built-in constraints we assume that a first order decidable theory $\mathcal{CT}$ is given which describes their meaning. Often the terminology "host language" is used to indicate the language consisting of the built-in predicates, because indeed often CHR is implemented on top of such an existing host language.

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 $\mathtt{chr}(c\#i) = c$ and $\mathtt{id}(c\#i) = i$, possibly extended to sets and sequences of identified CHR constraints in the obvious way.

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, and 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:

$$r \ @ \ H^k \setminus H^h \iff g \mid B$$

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 conjunction of built-in constraints and B is a 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. When the guard g is empty or *true* we omit $g \mid$. The names of rules are omitted when not needed. For a simplification rule we omit $H^k \setminus$ while we write a propagation rule as $H^k \Longrightarrow g \mid B$. A CHR *goal* is a multi-set of (both user-defined and built-in) constraints.

We also use the following 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 $\exists_{-V} \phi$ 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 and $t\sigma$ the application of a substitution σ to a syntactic object t.

CHR dialects. As mentioned before, the computational power of CHR depends on several aspects, including the number of atoms allowed in the heads, the underlying signature Σ on which programs are defined, and the constraint theory $\mathcal{CT}$, defining the built-ins. We use the notation CHR(X), where the parameter X indicates the signature and the constraint theory (in other words, the host language).

More precisely, the language under consideration in this paper is CHR(C) and has been defined in the introduction. We will also use the notation CHR(P) to denote *propositional* CHR, that is the language where all constraints have arity zero. This corresponds to consider a trivial host language without any data type. Finally CHR(F) indicates the (usual) CHR language which allows functor symbols and the $=$ built-in. Thus in this case the host language allows arbitrary Herbrand terms and supports unification among them.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

Solve $\langle \{c\} \uplus G, S, B, T\rangle_n \stackrel{\omega_t}{\rightarrow}_P \langle G, S, c \wedge B, T\rangle_n$ where c is a built-in constraint
Introduce $\langle \{c\} \uplus G, S, B, T\rangle_n \stackrel{\omega_t}{\rightarrow}_P \langle G, \{c\#n\} \cup S, B, T\rangle_{n+1}$ where c is a CHR constraint
Apply $\langle G, H_1 \cup H_2 \cup S, B, T\rangle_n \stackrel{\omega_t}{\rightarrow}_P \langle C \uplus G, H_1 \cup S, \theta \wedge B, T \cup \{t\}\rangle_n$ 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. $\mathtt{chr}(H_1) = H_1'\theta$, $\mathtt{chr}(H_2) = H_2'\theta$, $\mathcal{CT} \models B \rightarrow \exists_{-Fv(B)}(\theta \wedge g)$
and $t = \mathtt{id}(H_1) +\!\!+ \mathtt{id}(H_2) +\!\!+ [r] \notin T$

Table 1. *Transitions of* ω_t

The number of atoms in the heads also affects the expressive power of the language. We use the notation CHR_1, possibly combined with the notation above, to denote *single-headed* CHR, where heads of rules contain one atom.

Operational semantics of CHR. We consider the theoretical operational semantics, denoted by ω_t and the abstract semantics, denoted by ω_o. The semantics ω_t is given by Duck *et al.* (2004) as a state transition system $T = (Conf, \stackrel{\omega_t}{\rightarrow}_P)$ 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 $\stackrel{\omega_t}{\rightarrow}_P \subseteq Conf \times Conf$ 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) using rules from the program. The **Apply** transition is applicable when the current built-in store (B) entails the guard of the rule (g).

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. A computation is called *terminating* if it ends in a final configuration, *infinite* otherwise.

The first CHR operational semantics defined in (Frühwirth 1998) differs from the traditional semantics ω_t. Indeed this original, so called, abstract semantics denoted by ω_o, allows the firing of a propagation rule an infinite number of times. For this reason ω_o can be seen as the abstraction of the traditional semantics where the propagation history is not considered. It is identical to ω_t, except that configurations are of the form $\langle G, S, B\rangle_n$ (they do not contain a propagation history) and the **Apply** transition does not have the last condition that $t \notin T$.

3 Range-restricted CHR(C)

In this section we consider the (multi-headed) range-restricted CHR(C) language described in the introduction. We call a CHR rule range-restricted if all the variables

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

which appear in the body and in the guard appear also in the head of a rule. More formally, if $Var(X)$ denotes the variables used in X, the rule $r\ @H^k \backslash H^h \Longleftrightarrow g \mid B$ is range-restricted if $Var(B) \cup Var(g) \subseteq Var(H^k \backslash H^h)$ holds. A CHR language is called range-restricted if it allows range-restricted rules only.

We prove that in range-restricted CHR(C) the existence of an infinite computation is a decidable property when considering the ω_o semantics. This shows that this language is less expressive than Turing Machines and than CHR(C). Our result is based on the theory of well-structured transition systems (WSTS) and we refer to (Finkel and Schnoebelen 2001; Abdulla *et al.* 1996) for this theory. Here we only provide the basic definitions on WSTS, taken from (Finkel and Schnoebelen 2001).

Recall that a *quasi-order* (or, equivalently, preorder) is a reflexive and transitive relation. A *well-quasi-order* (wqo) is defined as a quasi-order $\leqslant$ over a set X such that, for any infinite sequence $x_0, x_1, x_2, \ldots$ in X, there exist indexes $i < j$ such that $x_i \leqslant x_j$.

A *transition system* is defined as usual, namely it is a structure $TS = (S, \rightarrow)$, where S is a set of *states* and $\rightarrow \subseteq S \times S$ is a set of *transitions*. We define $Succ(s)$ as the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of s. We say that TS is *finitely branching* if, for each $s \in S$, $Succ(s)$ is finite. Hence we have the key definition.

Definition 3.1 (*Well-structured transition system with strong compatibility*)
A *well-structured transition system with strong compatibility* is a transition system $TS = (S, \rightarrow)$, equipped with a quasi-order $\leqslant$ on S, such that the two following conditions hold:

1. $\leqslant$ is a well-quasi-order;
2. $\leqslant$ is strongly (upward) compatible with $\rightarrow$, that is, for all $s_1 \leqslant t_1$ and all transitions $s_1 \rightarrow s_2$, there exists a state t_2 such that $t_1 \rightarrow t_2$ and $s_2 \leqslant t_2$ holds.

The next theorem is a special case of a result in (Finkel and Schnoebelen 2001) and will be used to obtain our decidability result.

Theorem 3.2
Let $TS = (S, \rightarrow, \leqslant)$ be a finitely branching, well-structured transition system with strong compatibility, decidable $\leqslant$ and computable $Succ(s)$ for $s \in S$. Then the existence of an infinite computation starting from a state $s \in S$ is decidable.

Decidability of divergence. Consider a given goal G and a (CHR) program P and consider the transition system $T = (Conf, \overset{\omega_o}{\rightarrow}_P)$ defined in Section 2. Obviously the number of constants and variables appearing in G or in P is finite. Moreover, observe that since we consider range-restricted programs, the application of the transitions $\overset{\omega_o}{\rightarrow}_P$ does not introduce new variables in the computations. In fact, even though rules are renamed (in order to avoid clash of variables), the definition of the Apply rule (in particular the definition of θ) implies that in a transition $s_1 \overset{\omega_o}{\rightarrow}_P s_2$ we have that $Var(s_2) \subseteq Var(s_1)$ holds. Hence an obvious inductive argument implies that no new variables arise in computations. For this reason, given a goal G and a program P, we can assume that the set $Conf$ of all the configurations uses only a finite number of constants and variables. In the following we implicitly make this assumption. We define a quasi-order on configurations as follows.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

Definition 3.3
Given two configurations $s_1 = \langle G_1, S_1, B_1 \rangle_i$ and $s_2 = \langle G_2, S_2, B_2 \rangle_j$ we say that $s_1 \leqslant s_2$ if

- for every constraint $c \in G_1$ $|\{c \in G_1\}| \leqslant |\{c \in G_2\}|$
- for every constraint $c \in \{d \,.\, d\#i \in S_1\}$ $|\{i \,.\, c\#i \in S_1\}| \leqslant |\{i \,.\, c\#i \in S_2\}|$
- B_1 is logically equivalent to B_2

The next Lemma, with proof in (Gabbrielli *et al.* 2010), states the relevant property of $\leqslant$.

Lemma 3.4
$\leqslant$ is a well-quasi-order on $Conf$.

Next, in order to obtain our decidability results we have to show that the strong compatibility property holds. This is the content of the following lemma whose proof is in Gabbrielli *et al.* (2010).

Lemma 3.5
Given a CHR(C) program P, $(Conf, \stackrel{\omega_o}{\rightarrow}_P, \leqslant)$ is a well-structured transition system with strong compatibility.

Finally we have the desired result.

Theorem 3.6
Given a range-restricted CHR(C) program P and a goal G, the existence of an infinite computation for G in P is decidable.

Proof
First observe that, due to our assumption on range-restricted programs, $T = (Conf, \stackrel{\omega_o}{\rightarrow}_P)$ is finitely branching. In fact, as previously mentioned, the use of rule Apply can not introduce new variables (and hence new different states). The thesis follows immediately from Lemma 3.5 and Theorem 3.2. $\square$

The previous Theorem implies that range-restricted CHR(C) is strictly less expressive than Turing Machines, in the sense that there can not exist a termination preserving encoding of Turing Machines into range-restricted CHR(C). To be more precise, we consider an encoding of a Turing Machine into a CHR language as a function f which, given a machine Z and an initial instantaneous description D for Z, produces a CHR program and a goal. This is denoted by $(P, G) = f(Z, D)$. Hence we have the following.

Definition 3.7 (*Termination preserving encoding*)
An encoding f of Turing Machines into a CHR language is termination preserving[2] if the following holds: the machine Z starting with D terminates iff the goal G in the CHR program P has only terminating computations, where $(P, G) = f(Z, D)$.

[2] For many authors the existence of a termination preserving encoding into a non-deterministic language L is equivalent to the Turing completeness of L, however there is no general agreement on this, since for others a weak termination preserving encoding suffices.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

The encoding is weak termination preserving if: the machine Z starting with D terminates iff the goal G in the CHR program P has at least one terminating computation.

Since termination is undecidable for Turing Machines, we have the following immediate corollary of Theorem 3.6.

Corollary 3.8
There exists no termination preserving encoding of Turing Machines into range-restricted CHR(C).

Note that the previous result does not exclude the existence of weak encodings. For example, in (Busi *et al.* 2004) it is showed that the existence an infinite computation is decidable in CCS!, a variant of CCS, yet it is possible to provide a weak termination preserving encoding of Turing Machines in CCS! (essentially by adding spurious non-terminating computations). We conjecture that such an encoding is not possible for CHR(C). Note also that previous results imply that range-restricted CHR(C) is strictly less expressive than CHR(C): in fact there exists a termination preserving encoding of Turing Machines into CHR(C) (Sneyers 2008; Di Giusto *et al.* 2009).

4 Single-headed CHR(C)

As mentioned in the introduction, while CHR(C) and $\text{CHR}_1(F)$ are Turing complete languages (Sneyers 2008; Di Giusto *et al.* 2009), the question of the expressive power of $\text{CHR}_1(C)$ is open. Here we answer to this question by proving that the existence of a terminating computation is decidable for this language, thus showing that $\text{CHR}_1(C)$ is less expressive than Turing machines. Throughout this section, we assume that the abstract semantics ω_o is considered (however see the discussion at the end for an extension to the case of ω_t). The proof we provide is a direct one, since neither well-structured transition systems nor reduction to Petri Nets can be used here (see the introduction).

4.1 Some preparatory results

We introduce here two more notions, namely the forest associated to a computation and the notion of reactive sequence, and some related results. We will need them for the main result of this section.

First, we observe that it is possible to associate to the computation for an atomic goal G in a program P a tree where, intuitively, nodes are labeled by constraints (recall that these are atomic formulae), the root is G and every child node is obtained from the parent node by firing a rule in the program P. This notion is defined precisely in the following, where we generalize it to the case of a generic (non atomic) goal, where for each CHR constraint in the goal we have a tree. Thus we obtain a *forest* $F_\delta = (V, E)$ associated to a computation δ, where V contains a node for each repetition of identified CHR constraints in δ. Before defining the forest we need the concept of *repetition* of an identified CHR atom in a computation.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

Definition 4.1 (*Repetition*)
Let P be a CHR program and let δ be a computation in P. We say that an occurrence of an identified CHR constraint $h\#l$ in δ is the i-th repetition of $h\#l$, denoted by $h\#l^i$, if it is preceded in δ by i **Apply** transitions of propagation rules whose heads match the atom $h\#l$. We also define

$$r(\delta, h\#l) = max\{i \mid \text{ there exists a } i\text{-th repetition of } h\#l \text{ in } \delta\}$$

Definition 4.2 (*Forest*)
Let δ be a terminating computation for a goal in a $\mathrm{CHR}_1(C)$ program. The forest associated to δ, denoted by $F_\delta = (V, E)$ is defined as follows. V contains nodes labeled either by repetitions of identified CHR constraints in δ or by $\Box$. E is the set of edges. The labeling and the edges in E are defined as follows:
(a) For each CHR constraint k which occurs in the first configuration of δ there exists a tree in $F_\delta = (V, E)$, whose root is labeled by a repetition $k\#l^0$, where $k\#l$ is the identified CHR constraint associated to k in δ.
(b) If n is a node in $F_\delta = (V, E)$ labeled by $k\#l^i$ and the rule $r\ @h \odot g \mid C, k_1, \dots, k_m$ is used in δ to rewrite the repetition $h\#l^i$, where $\odot \in \{\Longleftrightarrow, \Longrightarrow\}$, the k_i's are CHR constraints while C contains built-ins, then we have two cases:

1. If $\odot$ is $\Longrightarrow$ then n has $m+1$ sons, labeled by $k_j\#{l_j}^0$, for $j \in [1, m]$, and by $h\#l^{i+1}$, where the $k_j\#{l_j}^0$ are the repetitions generated by the application of the rule r to $h\#l^i$ in δ.
2. If $\odot$ is $\Longleftrightarrow$ then:
   - if $m > 0$ then n has m sons, labeled by $k_j\#{l_j}^0$, for $j \in [1, m]$, where $k_j\#{l_j}^0$ are the repetitions generated by the application of the rule r to $h\#l^i$ in δ.
   - if $m = 0$ then n has 1 son, labeled by $\Box$.

Note that, according to the previous definition, nodes which are not leaves are labeled by repetitions of identified constraints $k\#l^i$, where either $i < r(\delta, h\#l)$ or $h\#l$ does not occur in the last configuration of δ. On the other hand, the leaves of the trees in F_δ are labeled either by $\Box$ or by the repetitions which do not satisfy the condition above. An example can help to understand this crucial definition.

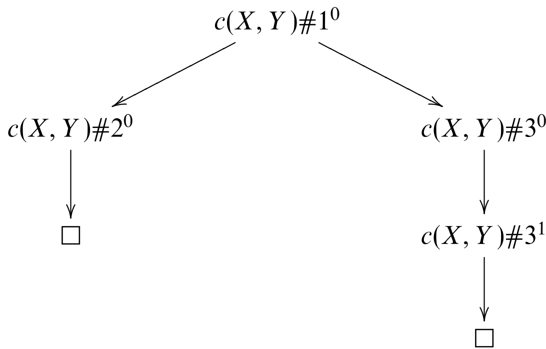
Example 4.3
Let us consider the following program P:

```
r1 @ c(X,Y) <=> c(X,Y),c(X,Y)
r2 @ c(X,Y) <=> X = 0
r3 @ c(0,Y) ==> Y = 0
r4 @ c(0,0) <=> true
```

There exists a terminating computation δ for the goal $c(X, Y)$ in the program P, which uses the clauses r1, r2, r3, r4 in that order and whose associated forest F_δ is

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

the following tree:



Note that the left branch corresponds to the termination obtained by using rule `r2`, hence the superscript is not incremented. On the other hand, in the right branch the superscript ^{0} at the second level becomes ^{1} at the third level. This indicates that a propagation rule (rule `r3`) has been applied.

Given a forest F_δ, we write $T_\delta(n)$ to denote the subtree of F_δ rooted in the node n. Moreover, we identify a node with its label and we omit the specification of the repetition, when not needed. The following definition introduces some further terminology that we will need later.

Definition 4.4

- Given a forest F_δ, a path from a root of a tree in the forest to a leaf is called a *single constraint computation*, or *sc-computation* for short.
- Two repetitions $h\#l^i$ and $k\#m^j$ of identified CHR constraints are called r-equal, indicated by $h\#l^i == k\#m^j$, iff there exists a renaming ρ such that $h = k\rho$.
- a sc-computation σ is p-repetitive if $p = \max_{h\#l^i \in \sigma} |\{k\#m^j \in \sigma \mid \ h\#l^i == k\#m^j\}|$.
- The degree of a p-repetitive sc-computation σ, denoted by $dg(\sigma)$ is the cardinality of the set P_REP which is defined as the maximal set having the following properties:
  - contains a repetition $h\#l^i$ in σ iff $p = |\{k\#m^j \in \sigma \mid \ h\#l^i == k\#m^j\}|$
  - if $h\#l^i$ is in P_REP then P_REP does not contain a repetition $k\#m^j$ s.t. $h\#l^i == k\#m^j$
- A forest F_δ is l-repetitive if one of its sc-computation σ is l-repetitive and there is no l'-repetitive sc-computation σ' in F_δ with $l' > l$.
- The degree $dg(F_\delta)$ of an l-repetitive forest F_δ is defined as

$$dg(F_\delta) = \sum_\sigma \{dg(\sigma) \mid \ \sigma \text{ is an } l\text{-repetitive sc-computation in } F_\delta\}.$$

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

After the forest, the second main notion that we need to introduce is that one of reactive sequence[3].

Given a computation δ, we associate to each (repetition of an) occurrence of an identified CHR atom $k\#l$ in δ a, so called, reactive sequence of the form $\langle c_1, d_1\rangle \dots \langle c_n, d_n\rangle$, where, for any $i \in [1, n]$, c_i, d_i are built-in constraints.

Intuitively each pair $\langle c_i, d_i\rangle$ of built-in constraints represents all the **Apply** transition steps, in the computation δ, which are used to rewrite the considered occurrence of the identified CHR atom $k\#l$ and the identified atoms derived from it. The constraint c_i represents the input for this sequence of **Apply** computation steps, while d_i represents the output of such a sequence. Hence one can also read such a pair as follows: the identified CHR constraint $k\#l$, in δ, can transform the built-in store from c_i to d_i. Different pairs $\langle c_i, d_i\rangle$ and $\langle c_j, d_j\rangle$ in the reactive sequence correspond to different sequences of **Apply** transition steps. This intuitive notion is further clarified later (Definition 4.9), when we will consider a reactive sequence associated to a repetition of an identified CHR atom.

Since in CHR computations the built-in store evolves monotonically, i.e. once a constraint is added it can not be retracted, it is natural to assume that reactive sequences are monotonically increasing. So in the following we will assume that, for each reactive sequence $\langle c_1, d_1\rangle \dots \langle c_n, d_n\rangle$, the following condition holds: $CT \models d_j \rightarrow c_j$ and $CT \models c_{i+1} \rightarrow d_i$ for $j \in [1, n]$, $i \in [1, n-1]$. Moreover, we denote the empty sequence by ε. Next, we define the strictly increasing reactive sequences w.r.t. a set of variables X.

Definition 4.5 (*Strictly increasing sequence*)
Given a reactive sequence $s = \langle c_1, d_1\rangle \cdots \langle c_n, d_n\rangle$, with $n \geqslant 0$ and a set of variables X, we say that s is strictly increasing with respect to X if the following holds for any $j \in [1, n]$, $i \in [1, n-1]$

- $Fv(c_j, d_j) \subseteq X$,
- $CT \models d_i \not\rightarrow c_{i+1}$ and $CT \models c_i \not\rightarrow d_i$.

Given a generic reactive sequence $s = \langle c_1, d_1\rangle \cdots \langle c_n, d_n\rangle$ and a set of variables X, we can construct a new, strictly increasing sequence $\eta(s, X)$ with respect to a set of variables X as follows. First the operator η restricts all the constraints in s to the variables in X (by considering the existential closure with the exception of the variables in X). Then η removes from the sequence all the stuttering steps (namely the pairs of constraints $\langle c, d\rangle$, such that $CT \models c \leftrightarrow d$) except the last. Finally, in the sequence produced by the two previous steps, if there exists a pair of consecutive elements $\langle c_l, d_l\rangle\langle c_{l+1}, d_{l+1}\rangle$ which are "connected", in the sense that c_{l+1} does not provide more information than d_l, then such a pair is "fused" in (i.e., replaced by) the unique element $\langle c_l, d_{l+1}\rangle$ (and this is repeated inductively for the new pairs). This is made precise by the following definition.

[3] This notion is similar to that one used in the (trace) semantics of concurrent languages, see, for example, (de Boer and Palamidessi 1990; de Boer *et al.* 2000) for the case of concurrent constraint programming. The name comes from this field.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

Definition 4.6 (*Operator η*)
Let $s = \langle c_1, d_1\rangle \cdots \langle c_n, d_n\rangle$ be a sequence of pairs of built-in stores and let X be a set of variables. The sequence $\eta(s, X)$ is the obtained as follows:

1 First we define $s' = \langle c'_1, d'_1\rangle \cdots \langle c'_n, d'_n\rangle$, where for $j \in [1, n]$ $c'_j = \exists_{-X} c_j$ and $d'_j = \exists_{-X} d_j$.
2 Then we define s'' as the sequence obtained from s' by removing each pair of the form $\langle c, d\rangle$ such that $CT \models c \leftrightarrow d$, if such a pair is not the last one of the sequence.
3 Finally we define $\eta(s, X) = s'''$, where s''' is the closure of s'' w.r.t. the following operation: if $\langle c_l, d_l\rangle\langle c_{l+1}, d_{l+1}\rangle$ is a pair of consecutive elements in the sequence and $CT \models d_l \rightarrow c_{l+1}$ holds then such a pair is substituted by $\langle c_l, d_{l+1}\rangle$.

The following Lemma states a first useful property. The proof is in (Gabbrielli *et al.* 2010).

Lemma 4.7
Let X be a finite set of variables and let $s = \langle c_1, c_2\rangle \cdots \langle c_{n-1}, c_n\rangle$ be a strictly increasing sequence with respect to X. Then $n \leqslant |X| + 2$.

Next we note that, given a set of variables X the possible strictly increasing sequences w.r.t. X are finite (up to logical equivalence on constraints), if the set of the constants is finite. This is the content of the following lemma, whose proof is in (Gabbrielli *et al.* 2010). Here and in the following, with a slight abuse of notation, given two reactive sequences $s = \langle c_1, d_1\rangle \cdots \langle c_n, d_n\rangle$ and $s' = \langle c'_1, d'_1\rangle \cdots \langle c'_n, d'_n\rangle$, we say that s and s' are equal (up to logical equivalence) and we write $s = s'$, if for each $i \in [1, n]$ $CT \models c_i \leftrightarrow c'_i$ and $CT \models d_i \leftrightarrow d'_i$ holds.

Lemma 4.8
Let $Const$ be a finite set of constants and let S be a finite set of variables such that $u = |Const|$ and $w = |S|$. The set of sequences s which are strictly increasing with respect to S (up to logical equivalence) is finite and has cardinality at the most

$$\frac{2^{w(u+w)(w+3)} - 1}{2^{w(u+w)} - 1}.$$

Finally, we show how reactive sequences can be obtained from a forest associated to a computation. First we need to define the reactive sequence associated to a repetition of an identified CHR atom in a computation. In this definition we use the operator η introduced in Definition 4.6.

Definition 4.9
Let δ be a computation for a $\text{CHR}_1(C)$ program, $h\#l^j$ be a repetition of an identified CHR atom in δ and $r_1, \ldots, r_n$ the sequence of the **Apply** transition in δ that rewrite $h\#l^j$ and all the repetitions derived from it. If $s \xrightarrow{r_i}_P s'$ let $pair(r_i)$ be the pair $(\bigwedge B_1, \bigwedge B_2)$ where B_1 and B_2 are all the built-ins in s and s'. We will denote with $seq(h\#l^j, \delta)$ the sequence $\eta(pair(r_1) \ldots pair(r_n), Fv(h))$

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

Finally we define the function S_{F_δ} which, given a node n in a forest associated to a computation δ (see Definition 4.2), returns a reactive sequence. Such a sequence intuitively represents the sequence of the **Apply** transition steps which have been used in δ to rewrite the repetition labeling n and the repetitions derived from it.

Definition 4.10 (*Sequence associated to a node in a forest*)
Let δ be a terminating computation and let $F_\delta = (V, E)$ be the forest associated to it. Given a node n in F_δ we define:

- if the label of n is $h\#l^i$, then $S_{F_\delta}(n) = seq(h\#l^i, \delta)$;
- if the label of n is $\Box$ then $S_{F_\delta}(n) = \varepsilon$.

Example 4.11
Let us consider for instance the forest shown in Example 4.3. The sequences associated to the nodes of this forest are:

- $S_{F(\delta)}(c(X, Y)\#1^0) = \langle true, X = 0 \wedge Y = 0\rangle$
- $S_{F(\delta)}(c(X, Y)\#2^0) = \langle true, X = 0\rangle$
- $S_{F(\delta)}(c(X, Y)\#3^0) = \langle X = 0, X = 0 \wedge Y = 0\rangle$
- $S_{F(\delta)}(c(X, Y)\#3^1) = \langle X = 0 \wedge Y = 0, X = 0 \wedge Y = 0\rangle$

4.2 Decidability of termination

We are now ready to prove the main result of the paper. First we need the following Lemma which has some similarities to the pumping lemma of regular and context free grammars. Indeed, if the derivation is seen as a forest, this lemma allows us to compress a tree if in a path of the tree there are two r-equal constraints with an equal (up to renaming) sequence. The lemma is proved in (Gabbrielli *et al.* 2010).

Here and in the following given a node n in a forest F we denote by $A_F(n)$ the label associated to n.

Lemma 4.12
Let δ be a terminating computation for the goal G in the $\mathrm{CHR}_1(C)$ program P. Assume that F_δ is l-repetitive with $p = dg(F_\delta)$ and assume that there exists an l-repetitive sc-computation σ of F_δ and a repetition $k\#l^i \in \sigma$ such that $l = |\{h\#n^j \in \sigma \mid \; h\#n^j == k\#l^i\}|$.
Moreover assume that there exist two distinct nodes n and n' in σ such that n' is a node in $T_\delta(n)$, $A_{F_\delta}(n) = k\#l^i$, $A_{F_\delta}(n') = k'\#l'^{i'}$ and ρ is a renaming such that $S_{F_\delta}(n) = S_{F_\delta}(n')\rho$ and $k = k'\rho$.

Then there exists a terminating computation δ' for the goal G in the program P, such that either $F_{\delta'}$ is l'-repetitive with $l' < l$, or $F_{\delta'}$ is l-repetitive and $dg(F'_\delta) < p$.

Finally we obtain the following result, which is the main result of this paper.

Theorem 4.13 (*Decidability of termination*)
Let P be a $\mathrm{CHR}_1(C)$ program an let G be a goal. Let u be the number of distinct constants used in P and in G and let w be the maximal arity of the CHR constraints which occur in P and in G.

G has a terminating computation in P if and only if there exists a terminating computation δ for G in P s.t. F_δ is m-repetitive and $m \leqslant \frac{2^{w(u+w)(w+3)}-1}{2^{w(u+w)}-1} = L$.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

Proof
We prove only that if G has a terminating computation in P then there exists a terminating computation δ for G in P s.t. F_δ is m-repetitive and $m \leqslant L$. The proof of the converse is straightforward and hence it is omitted.

The proof is by contradiction. Assume G has a terminating computation δ in P s.t. F_δ is m-repetitive, $m > L$ and there is no terminating computation δ' for G in P such that $F_{\delta'}$ is m'-repetitive and $m' < m$. Moreover, without loss of generality, we can assume that the degree of F_δ is minimal, namely there is no terminating computation δ' for G in P such that $F_{\delta'}$ is m-repetitive and $dg(F_{\delta'}) < dg(F_\delta)$.

Let σ be a m-repetitive sc-computation in F_δ. By definition, there exist m repetitions of identified CHR constraints $k_1\#{l_1}^{i_1}, ..., k_r\#{l_m}^{i_m}$ in σ, which are r-equal. Therefore there exist renamings $\rho_{s,t}$ such that $k_s = k_t\rho_{s,t}$ for each $s, t \in [1, m]$.

By Lemma 4.8 for each CHR constraint k which occurs in P or in G, the set of sequences s which are strictly increasing with respect to $Fv(k)$ (up to logical equivalence) is finite and has cardinality at the most L. Then there are two distinct nodes n and n' in σ and there exist $s, t \in [1, m]$ such that $A(n) = k_s\#{l_s}^{i_s}$ and $A(n') = k_t\#{l_t}^{i_t}$ and $S_{F_\delta}(n) = S_{F_\delta}(n')\rho_{s,t}$. Then we have a contradiction, since by Lemma 4.12 this implies that there exists a terminating computation δ' for G in P s.t. either $F_{\delta'}$ is m'-repetitive with $m' < m$ or $F_{\delta'}$ is m-repetitive and $dg(F_{\delta'}) < dg(F_\delta)$ and then the thesis. □

As an immediate corollary of the previous theorem we have that the existence of a terminating computation for a goal G in a $\mathrm{CHR}_1(C)$ program P is decidable. Then we have also the following result, which is stronger than Corollary 3.8 since here weak encodings are considered.

Corollary 4.14
There is no weak termination preserving encoding of Turing Machines into $\mathrm{CHR}_1(C)$.

As mentioned at the beginning of this section, the previous result is obtained when considering the abstract semantics ω_o. However it holds also when considering the theoretical semantics ω_t. In fact Lemma 4.12 holds if we require that two r-equal constraints have the same sequence and have fired the same propagation rules. Since the propagation rules are finite Theorem 4.13 is still valid if $m \leqslant 2^r \cdot \frac{2^{w(u+w)(w+3)}-1}{2^{w(u+w)}-1}$ where r is the number of propagation rules.

5 Conclusions

We have shown two decidability results for two fragments of $\mathrm{CHR}(C)$, the CHR language defined over a signature which does not allow function symbols. The first result, in Section 3, assumes the abstract operational semantics, while the second one, in Section 4, holds for both semantics (abstract and theoretical). These results are not immediate. Indeed, $\mathrm{CHR}(C)$, without further restrictions and with any of the two semantics, is a Turing complete language (Sneyers 2008; Di Giusto *et al.* 2009). It remains quite expressive also with our restrictions: for example, $\mathrm{CHR}_1(C)$, the second fragment that we have considered, allows an infinite number of different states, hence, for example, it can not be translated to Petri Nets.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

Host language X	*Operational semantics*	$k = 1$	$k > 1$
P (propositional)	abstract	No	No
range-restricted C (constants) (cf. Section 3)	abstract	No	**No**
C (constants), without =	any	No	Yes
C (constants) (cf. Section 4)	any	**No**	Yes
F (functors)	any	Yes	Yes

Table 2. *Termination preserving encoding of Turing Machines into* $CHR_k(X)$

These results imply that range-restricted CHR(C) and CHR$_1$(C), the two considered fragments, are strictly less expressive than Turing Machines (and therefore than CHR(C)). Also, it seems that range-restricted CHR(C) is more expressive that CHR$_1$(C), since the decidability result for the second language is stronger. However, a direct result in this sense is left for future work. Also, we leave to future work to establish a decidability result for range-restricted CHR(C) under an operational semantics which includes a propagation history. This is not easy, since in this case it appears difficult to apply the theory of well-structured transition systems (the well-quasi-order we have defined does not work).

Several papers have considered the expressive power of CHR in the last few years. In particular, Sneyers (2008) showed that a further restriction of CHR$_1$(C), which does not allow built-ins in the body of rules (and which therefore does not allow unification of terms) is not Turing complete. This result is obtained by translating CHR$_1$(C) programs (without unification) into propositional CHR and using the encoding of propositional CHR intro Petri Nets provided in (Betz 2007). The translation to propositional CHR is not possible for the language (with unification) CHR$_1$(C) that we consider. Betz (2007) also provides a translation of range-restricted CHR(C) to Petri nets. However in this translation, differently from our case, it is also assumed that no unification built-in can be used in the rules, and only ground goals are considered. Related to this paper is also (Di Giusto *et al.* 2009), where it is shown that CHR(F) is Turing complete and that restricting to single-headed rules decreases the computational power of CHR. However, these results are based on the theory of language embedding, developed in the field of concurrency theory to compare Turing complete languages, hence they do not establish any decidability result. Another related study 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. Earlier works by Frühwirth (Frühwirth and Abdennadher 2001; Frühwirth 2002) 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 clearly different from ours.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

A summary of the existing results concerning the computational power of several dialects of CHR is shown in Table 2. In this table, “no” and “yes” refer to the existence of a termination preserving encoding of Turing Machines into the considered language, while “any” means theoretical or abstract. The new results shown in this paper are indicated in a bold font.

Acknowledgements

We would like to thank the reviewers for their precise and helpful comments.

References

Abdulla, P. A., Cerans, K., Jonsson, B., and Tsay, Y.-K. 1996. General decidability theorems for infinite-state systems. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*. IEEE, 313–321.

Betz, H. 2007. Relating coloured Petri nets to Constraint Handling Rules. In *4th Workshop on Constraint Handling Rules*, K. Djelloul, G. J. Duck, and M. Sulzmann, Eds. Porto, Portugal, 33–47.

Busi, N., Gabbrielli, M., and Zavattaro, G. 2004. Comparing recursion, replication, and iteration in process calculi. In *ICALP*, J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, Eds. Lecture Notes in Computer Science, vol. 3142. Springer, 307–319.

de Boer, F. S., Gabbrielli, M., and Meo, M. C. 2000. A timed concurrent constraint language. *Information and Computation 161,* 1, 45–83.

de Boer, F. S. and Palamidessi, C. 1990. On the asynchronous nature of communication in concurrent logic languages: A fully abstract model based on sequences. In *CONCUR*, J. C. M. Baeten and J. W. Klop, Eds. Lecture Notes in Computer Science, vol. 458. Springer, 99–114.

Di Giusto, C., Gabbrielli, M., and Meo, M. C. 2009. Expressiveness of multiple heads in CHR. In *SOFSEM*, M. Nielsen, A. Kucera, *et al.*, Eds. Lecture Notes in Computer Science, vol. 5404. Springer, 205–216.

Duck, G. J., Stuckey, P. J., García de la Banda, M., and Holzbaur, C. 2004. The refined operational semantics of Constraint Handling Rules. In *ICLP ’04*, B. Demoen and V. Lifschitz, Eds. Lecture Notes in Computer Science, vol. 3132. Springer, Saint-Malo, France, 90–104.

Finkel, A. and Schnoebelen, P. 2001. Well-structured transition systems everywhere! *Theoretical Computer Science 256,* 1–2, 63–92.

Frühwirth, T. 1998. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming, Special Issue on Constraint Logic Programming 37,* 1–3, 95–138.

Frühwirth, T. 2009. *Constraint Handling Rules*. Cambridge University Press.

Frühwirth, T. W. 2002. As time goes by: Automatic complexity analysis of simplification rules. In *KR*, D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, Eds. Morgan Kaufmann, 547–557.

Frühwirth, T. W. and Abdennadher, S. 2001. The Munich rent advisor: A success for logic programming on the internet. *TPLP 1,* 3, 303–319.

Gabbrielli, M., Mauro, J., Meo, M. C., and Sneyers, J. 2010. *Decidability Properties for Fragments of CHR*. Technical report Available from `http://www.cs.unibo.it/~jmauro/papers/tech_report_iclp_2010`.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press

Sneyers, J. 2008. Turing-complete subclasses of CHR. In *ICLP*, M. G. de la Banda and E. Pontelli, Eds. Lecture Notes in Computer Science, vol. 5366. Springer, 759–763.

Sneyers, J., Schrijvers, T., and Demoen, B. 2009. The computational power and complexity of Constraint Handling Rules. *ACM Transactions on Programming Languages and Systems 31,* 2.

Sneyers, J., Van Weert, P., De Koninck, L., and Schrijvers, T. 2010. As time goes by: Constraint Handling Rules—A survey of CHR research between 1998 and 2007. *Theory and Practice of Logic Programming 10,* 1 (January), 1–47.

https://doi.org/10.1017/S1471068410000311 Published online by Cambridge University Press