On the expressive power of multiple heads in CHR

CINZIA DI GIUSTO, INRIA Rhône Alpes, Grenoble, France
MAURIZIO GABBRIELLI, Dipartimento di Scienze dell'Informazione and Lab. Focus INRIA,
Università di Bologna, Italy
MARIA CHIARA MEO, Dipartimento di Scienze, Università di Chieti Pescara, Italy

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. CHR programs consist of multi-headed guarded rules which allow to rewrite constraints into simpler ones until a solved form is reached. Many empirical evidences suggest that multiple heads augment the expressive power of the language, however no formal result in this direction has been proved, so far.

In the first part of this paper we analyze the Turing completeness of CHR with respect to the underlying constraint theory. We prove that if the constraint theory is powerful enough then restricting to single head rules does not affect the Turing completeness of the language. On the other hand, differently from the case of the multi-headed language, the single head CHR language is not Turing powerful when the underlying signature (for the constraint theory) does not contain function symbols.

In the second part we prove that, no matter which constraint theory is considered, under some reasonable assumptions it is not possible to encode the CHR language (with multi-headed rules) into a single headed language while preserving the semantics of the programs. We also show that, under some stronger assumptions, considering an increasing number of atoms in the head of a rule augments the expressive power of the language.

These results provide a formal proof for the claim that multiple heads augment the expressive power of the CHR language.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*Constraint and logic languages*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Concurrent programming structures*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Relations between models*; F.1.2 [**Computation by Abstract Devices**]: Models of Computation—*Parallelism and concurrency*; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*Control primitives*

General Terms: Languages, Theory

Additional Key Words and Phrases: CHR, expressiveness, language embedding, multiset rewriting systems

1. INTRODUCTION

Constraint Handling Rules (CHR) [Frühwirth 1991; Frühwirth 1998] 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 to rewrite constraints

into simpler ones until a solved form is reached. The language is parametric with respect to an underlying constraint theory CT which defines the meaning of basic built-in constraints.

The presence of multiple heads is a crucial feature which differentiates CHR from other existing committed choice (logic) languages. Many examples in the vast literature on CHR provide empirical evidence for the claim that such a feature is needed in order to obtain reasonably expressive constraint solvers in a reasonably simple way (see the discussion in [Frühwirth 1998]). However this claim was not supported by any formal result, so far.

In this paper we prove that multiple heads do indeed augment the expressive power of CHR. Since we know that CHR is Turing powerful [Sneyers et al. 2005], we first show that CHR with single heads, called CHR_1 in what follows, is also Turing powerful, provided that the underlying constraint theory allows the equality predicate (interpreted as pattern matching) and that the signature contains at least one function symbol (of arity greater than zero). This result is certainly not surprising; however it is worth noting that, as we prove later, when considering an underlying (constraint theory defined over a) signature containing finitely many constant symbols and no function symbol CHR (with multiple heads) is still Turing complete, while this is not the case for CHR_1.

This provides a first separation result which is however rather weak, since usual constraint theories used in CHR do allow function symbols. Moreover, in general computability theory is not always the right framework for comparing the expressive power of concurrent languages and several alternative formal tools have been proposed for this purpose. In fact, most concurrent languages are Turing powerful and nevertheless, because of distributed and concurrent actions, they can exhibit a quite different observational behaviour and expressive power. For example, a language with synchronous communication allows to solve a distributed problem which is unsolvable by using the asynchronous version of that language [Palamidessi 2003].

Hence, in the second part of the paper, we compare the expressive power of CHR and CHR_1 by using the notion of language encoding, first formalized in [de Boer and Palamidessi 1994; Shapiro 1989; Vaandrager 1993]. 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 behaviour 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 with respect to (some of) the syntactic operators of the language [de Boer and Palamidessi 1994].

We prove that CHR cannot be encoded into CHR_1 under the following three assumptions. First we assume that the observable properties to be preserved are the constraints computed by a program for a goal, more precisely we consider data sufficient answers and qualified answers. Since these are the two typical CHR observables for most CHR reference semantics, assuming their preservation is rather natural. Secondly we require that both the source CHR language and the target CHR_1 share the same constraint theory defining built-in constraints. This is also a natural assumption, as CHR programs are usually written to define a new (user-defined) predicate in terms of the existing built-in constraints. Finally we assume that the translation of a goal is compositional with respect to conjunction of goals, more precisely 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 one can translate parts of the goal separately. It is worth noticing that we do not impose any restriction on the translation of the program rules.

Finally, our third contribution shows that also the number of atoms (greater than one) affects the expressive power of the language. In fact we prove that, under some slightly stronger assumptions on the translation of goals, there exists no encoding of CHR_n (CHR with at most n atoms in the head of the rules) into CHR_m, for $m < n$.

The remainder of this paper is organized as follows. Section 2 introduces the languages under consideration. We then provide the encoding of two counters machines [Minsky 1967] in CHR_1 and discuss the Turing completeness of this language in Section 3. Section 4 contains the separation results for CHR and CHR_1 by considering first data sufficient answers and then qualified answers. In Section 5 we compare CHR_n and CHR_m, while Section 6 concludes by discussing related works and indicating some further development of this work. A shorter version of this paper, containing part of the results presented here, appeared in [Di Giusto et al. 2009].

2. PRELIMINARIES

In this section we give an overview of CHR syntax and operational semantics following [Frühwirth 1998].

2.1. CHR constraints and notation

We first need to distinguish the constraints handled by an existing solver, called built-in (or predefined) constraints, from those defined by the CHR program, called user-defined (or CHR) constraints. Therefore we assume that the signature Σ on which programs are defined contains two disjoint sets of predicate symbols Π_b for built-in and Π_u for user-defined constraints. In the following, as usual, an atomic constraint is a first-order atomic formula.

Definition 2.1 (*Built-in constraint*). A *built-in constraint* c is defined by:

$$c ::= a \mid c \land c \mid \exists_x c$$

where a is an atomic constraint which uses a predicate symbol from Π_b.

For built-in constraints we assume given a (first order) theory CT which describes their meaning.

Definition 2.2 (*User-defined constraint*). A *user-defined (or CHR) constraint* is an atomic constraint which uses a predicate symbols from Π_u.

We use c, d to denote built-in constraints, h, k to denote CHR constraints and a, b, f, g to denote both built-in and user-defined constraints (we will call these generally constraints). The capital versions of these notations will be used to denote multisets of constraints. We also denote by `false` any inconsistent conjunction of constraints and with `true` the empty multiset of built-in constraints.

We will use "," rather than $\land$ to denote conjunction and we will often consider a conjunction of atomic constraints as a multiset of atomic constraints: We prefer to use multisets rather than sequences (as in the original CHR papers) because our results do not depend on the order of atoms in the rules. In particular, we will use this notation based on multisets in the syntax of CHR.

The notation $\exists_V \phi$, where V is a set of variables, denotes the existential closure of a formula ϕ with respect to 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.

Moreover, if $\bar{t} = t_1, \ldots t_m$ and $\bar{t'} = t'_1, \ldots t'_m$ are sequences of terms then the notation $p(\bar{t}) = p'(\bar{t'})$ represents the set of equalities $t_1 = t'_1, \ldots, t_m = t'_m$ if $p = p'$, and it is undefined otherwise. This notation is extended in the expected way to multiset of constraints.

2.2. Syntax

A CHR program is defined as a set of two kinds of rules: simplification and propagation (some papers consider also simpagation rules, since these are abbreviations for propagation and simplification rules we do not need to introduce them). Intuitively, simplification

Table I. The standard transition system for CHR

Solve	$\dfrac{CT \models c \wedge d \leftrightarrow d' \text{ and c is a built-in constraint}}{\langle (c, G), K, d \rangle \longrightarrow \langle G, K, d' \rangle}$
Introduce	$\dfrac{\text{h is a user-defined constraint}}{\langle (h, G), K, d \rangle \longrightarrow \langle G, (h, K), d \rangle}$
Simplify	$\dfrac{H \Leftrightarrow C \mid B \in P \quad x = Fv(H) \quad CT \models d \rightarrow \exists_x ((H = H') \wedge C)}{\langle G, H' \wedge K, d \rangle \longrightarrow \langle B \wedge G, K, H = H' \wedge d \rangle}$
Propagate	$\dfrac{H \Rightarrow C \mid B \in P \quad x = Fv(H) \quad CT \models d \rightarrow \exists_x ((H = H') \wedge C)}{\langle G, H' \wedge K, d \rangle \longrightarrow \langle B \wedge G, H' \wedge K, H = H' \wedge d \rangle}$

rewrites constraints into simpler ones, while propagation adds new constraints which are logically redundant but may trigger further simplifications.

Definition 2.3. A CHR *simplification* rule has the form:

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

while a CHR *propagation* rule has the form:

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

where r is a unique identifier of a rule, H (the head) is a (non-empty) multiset of user-defined constraints, C (the guard) is a possibly empty multiset of built-in constraints and B (the body) is a possibly empty multiset of (built-in and user-defined) constraints.

A CHR *program* is a finite set of CHR simplification and propagation rules.

A CHR *goal* is a multiset of (both user-defined and built-in) constraints.

In the following, when the guard is `true` we omit `true|`. Also the identifiers of rules are omitted when not needed. An example of a CHR program is shown in next Section: Example 2.6.

2.3. Operational semantics

We describe now the operational semantics of CHR by slightly modifying the transition system defined in [Frühwirth 1998].

We use a transition system $T = (Conf, \longrightarrow)$ where configurations in $Conf$ are triples of the form $\langle G, K, d \rangle$, where G are the constraints that remain to be solved, K are the user-defined constraints that have been accumulated and d are the built-in constraints that have been simplified. An *initial configuration* has the form $\langle G, \emptyset, \emptyset \rangle$ while a *final configuration* has either the form $\langle G, K, \texttt{false} \rangle$ when it is *failed*, or the form $\langle \emptyset, K, d \rangle$, where d is consistent, when it is successfully terminated because there are no applicable rules.

Given a program P, the transition relation $\longrightarrow \subseteq Conf \times Conf$ is the least relation satisfying the rules in Table I (for the sake of simplicity, we omit indexing the relation with the name of the program). 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 transitions **Simplify** and **Propagate** allow 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, both these transitions assume that all variables appearing in a program clause are fresh ones. Both the **Simplify** and **Propagate** transitions are applicable when the current store (d) is strong enough to entail the guard of the rule (C), once the

parameter passing has been performed (this is expressed by the equation $H = H'$). Note that, due to the existential quantification over the variables x appearing in H, in such a parameter passing the information flow is from the actual parameters (in H') to the formal parameters (in H), that is, it is required that the constraints H' which have to be rewritten are an instance of the head H. This means that the equations $H = H'$ express pattern matching rather than unification. The difference between **Simplify** and **Propagate** lies in the fact that while the former transition removes the constraints H' which have been rewritten from the CHR constraint store, this is not the case for the latter.

Given a goal G, the operational semantics that we consider observes the final store of computations terminating with an empty goal and an empty user-defined constraint. We call these observables *data sufficient answers* slightly deviating from the terminology of [Frühwirth 1998] (a goal which has a data sufficient answer is called a data-sufficient goal in [Frühwirth 1998]).

Definition 2.4. [Data sufficient answers] Let P be a program and let G be a goal. The set $\mathcal{SA}_P(G)$ of data sufficient answers for the query G in the program P is defined as:

$$\mathcal{SA}_P(G) = \{\exists_{-Fv(G)} d \mid \langle G, \emptyset, \emptyset \rangle \longrightarrow^* \langle \emptyset, \emptyset, d \rangle \nrightarrow \}.$$

We also consider the following different notion of answer, obtained by computations terminating with a user-defined constraint which does not need to be empty.

Definition 2.5. [Qualified answers [Frühwirth 1998]] Let P be a program and let G be a goal. The set $\mathcal{QA}_P(G)$ of qualified answers for the query G in the program P is defined as:

$$\mathcal{QA}_P(G) = \{\exists_{-Fv(G)} (K \wedge d) \mid \langle G, \emptyset, \emptyset \rangle \longrightarrow^* \langle \emptyset, K, d \rangle \nrightarrow \}.$$

Both previous notions of observables characterize an input/output behaviour, since the input constraint is implicitly considered in the goal. Clearly in general $\mathcal{SA}_P(G) \subseteq \mathcal{QA}_P(G)$ holds, since data sufficient answers can be obtained by setting $K = \emptyset$ in Definition 2.5.

Note that in the presence of propagation rules, the naive operational semantics that we consider here introduces redundant infinite computations (because propagation rules do not remove user defined constraints). It is possible to define a different operational semantics (see [Abdennadher 1997]) which avoids these infinite computations by allowing to apply at most once a propagation rule to the same constraints. The results presented here hold also in case this semantics is considered, essentially because the number of applications of propagations rules does not matter. We refer here to the naive operational semantics because it is simpler than that one in [Abdennadher 1997].

An example can be useful to see what kind of programs we are considering here. The following program implements the sieve of Eratosthenes to compute primes.

Example 2.6. The following CHR program which consists of three simplifications rules, given a goal $upto(N)$ with N natural number, computes all prime numbers up to N: the first two rules generate all the possible candidates as prime numbers, while the third one removes all the incorrect information.

$$upto(1) \Leftrightarrow \mathtt{true}$$
$$upto(N) \Leftrightarrow N > 1 \mid prime(N), upto(N-1)$$
$$prime(X), prime(Y) \Leftrightarrow X \bmod Y = 0 \mid prime(Y).$$

For example suppose that the goal is $upto(4)$ then the following is one of the possible computations of the program where, by using the first two rules, from the goal $upto(4)$ we can generate all possible candidates,

$$\langle upto(4), \emptyset, \emptyset \rangle \longrightarrow^* \langle \emptyset, (prime(4), prime(3), prime(2)), \emptyset \rangle.$$

Then the third rule can be used to check, for every couple of constraints $prime(X)$, $prime(Y)$, if X is divisible by Y and in this case restores in the pool of constraints only the constraint $prime(Y)$ (in other words, we remove the constraint $prime(X)$). Thus we obtain:

$$\langle \emptyset, (prime(4), prime(3), prime(2)), \emptyset \rangle \longrightarrow^* \langle \emptyset, (prime(3), prime(2)), \emptyset \rangle.$$

Since there are no applicable rules $\langle \emptyset, (prime(3), prime(2)), \emptyset \rangle$ is a final configuration. Note that this is a qualified answer and the program with this goal has no data sufficient answers.

In the following, we study several CHR dialects defined by setting a limit in the number of the atoms present in the head of rules and by considering the possibility of non trivial data sufficient answers, as described by the following two definitions.

Definition 2.7. A data sufficient answer for the goal G in the program P is called trivial if it is equal to G (is called non trivial otherwise).

Definition 2.8 (*CHR dialects*). With CHR_n we denote a CHR language where the number of atoms in the head of the rules is at most n. Moreover, $\mathrm{CHR}_{n,d}$ denotes the language consisting of CHR_n programs which have (for some goal) non trivial data sufficient answers, while $\mathrm{CHR}_{n,t}$ denotes the language consisting of CHR_n programs which, for any goal, have only trivial data sufficient answers and qualified answers.

3. ON THE TURING COMPLETENESS OF CHR$_1$

In this section we discuss the Turing completeness of CHR_1 by taking into account also the underlying constraint theory CT and signature Σ (defined in the previous section). In order to show the Turing completeness of a language we encode two counter machines, also called Minsky machines, into it.

We recall here some basic notions on this Turing equivalent formalism. A two counter machine (2CM) [Minsky 1967] $M(v_0, v_1)$ consists of two registers R_1 and R_2 holding arbitrary large natural numbers and initialized with the values v_0 and v_1, and a program, i.e. a finite sequence of numbered instructions which modify the two registers. There are three types of instructions $j : Inst()$ where j is the number of the instruction:

— $j : Succ(R_i)$: adds 1 to the content of register R_i and goes to instruction $j + 1$;
— $j : DecJump(R_i, l)$: if the content of the register R_i is not zero, then decreases it by 1 and goes to instruction $j + 1$, otherwise jumps to instruction l;
— $j : Halt$: stops computation and returns the value in register R_1,

where $1 \leq i \leq 2$; $1 \leq j, l \leq n$ and n is the number of instructions of the program.

An internal state of the machine is given by a tuple (p_i, r_1, r_2) where the program counter p_i indicates the next instruction and r_1, r_2 are the current contents of the two registers. Given a program, its computation proceeds by executing the instructions as indicated by the program counter. The execution stops when the program counter reaches the *Halt* instruction.

As a first result, we show that CHR_1 is Turing powerful, provided that the underlying language signature Σ contains at least a function symbol (of arity one) and a constant symbol. This result is obtained by providing an encoding $[\![\ ]\!] : Machines \rightarrow \mathcal{P}_1$ of a two counter machine $M(v_0, v_1)$ in a CHR program ($\mathcal{P}_1$ denotes the set of CHR_1 programs) as shown in Figure 1: Every rule takes as input the program counter and the two registers and updates the state according to the instruction in the obvious way.

Note that, due to the pattern matching mechanism, a generic goal $i(p_i, s, t)$ can fire at most one of the two rules encoding the *DecJump* instruction (in fact, if s is a free variable no rule in the encoding of $p_i : DecJump(r_1, p_l)$ is fired).

$$\begin{aligned}
[\![p_i : Succ(r_1)]\!] &:= & i(p_i, R_1, R_2) &\Leftrightarrow i(p_{i+1}, succ(R_1), R_2) \\
[\![p_i : Succ(r_2)]\!] &:= & i(p_i, R_1, R_2) &\Leftrightarrow i(p_{i+1}, R_1, succ(R_2)) \\
[\![p_i : DecJump(r_1, p_l)]\!] &:= & i(p_i, 0, R_2) &\Leftrightarrow i(p_l, 0, R_2) \\
& & i(p_i, succ(R_1), R_2) &\Leftrightarrow i(p_{i+1}, R_1, R_2) \\
[\![p_i : DecJump(r_2, p_l)]\!] &:= & i(p_i, R_1, 0) &\Leftrightarrow i(p_l, R_1, 0) \\
& & i(p_i, R_1, succ(R_2)) &\Leftrightarrow i(p_{i+1}, R_1, R_2)
\end{aligned}$$

Fig. 1.   2CM encoding in CHR_1

Without loss of generality we can assume that the counters are initialized with 0, hence the encoding of a machine M with n instructions has the form:

$$[\![M(0,0)]\!] := \{[\![Instruction_1]\!], \ldots, [\![Instruction_n]\!]\}$$

(note that the initial values of the registers are not considered in the encoding of the machine: they will be used in the initial goal, as shown below). The following theorem, whose proof is immediate, states the correctness of the encoding (we use the notation $succ^k(0)$ to denote k applications of the functor $succ$ to 0).

THEOREM 3.1.   *A 2CM $M(0,0)$ halts with output $k > 0$ (or $k = 0$) on register R_1 if and only if the goal $i(1,0,0)$ in the program $[\![M(0,0)]\!]$ has a qualified answer $i(p_j, R_1', R_2')$, where $R_1' = succ^k(0)$ (or $R_1' = 0$).*

Note that the encoding provided in Figure 1 does not use any built-in, hence we can consider an empty theory CT here[1]. If the $=$ built-in is allowed in the body of rules then one could provide an encoding which gives the results of computation in terms of data sufficient answer, rather than qualified answer. To obtain this it is sufficient to add a fourth argument X (for the result) to the predicate i and to add the following translation for the *Halt* instruction:

$$[\![p_i : Halt]\!] := i(p_i, R_1, R_2, X) \Leftrightarrow X = R_1.$$

Such a translation in the previous encoding was not needed, since when one find the *Halt* instruction the CHR program simply stops and produces a qualified answer.

It is also worth noting that the presence of a function symbol ($succ()$ in our case) is crucial in order to encode natural numbers and therefore to obtain the above result. Indeed, when considering a signature containing only a finite number of constant symbols the language CHR_1, differently from the case of CHR, is not Turing powerful. To be more precise, assume that CT defines only the $=$ symbol, interpreted as pattern matching, which cannot be used in the body of rules (it can be used in the guards only). Assume also that the CHR language is now defined over a signature Σ containing finitely many constant symbols and no function symbol (of arity > 0). Let us call $CHR_\emptyset$ the resulting language.

As observed in [Sneyers 2008], $CHR_\emptyset$ (called in that paper single-headed CHR without host language) is computationally equivalent to propositional CHR (i.e. CHR with only zero-arity constraints), which can easily encoded into Petri nets. Since it is well known that in this formalism termination is decidable, we have the following result.

THEOREM 3.2.   *[Sneyers 2008] $CHR_\emptyset$ is not Turing complete.*

On the other hand, CHR (with multiple heads) is still Turing powerful also when considering a signature containing finitely many constant symbols and no function symbol, and

---

[1]We used the $=$ built-in the the operational semantics in order to perform parameter passing, however this is only a meta-notation which does not mean that the built-in $=$ has to be used in the language.

$$
\begin{aligned}
[\![p_i : Succ(r_1)]\!]_2 &:= & i(p_i, R_1, R_2) &\Leftrightarrow s(R_1, SuccR_1), i(p_{i+1}, SuccR_1, R_2) \\
[\![p_i : Succ(r_2)]\!]_2 &:= & i(p_i, R_1, R_2) &\Leftrightarrow s(R_2, SuccR_2), i(p_{i+1}, R_1, SuccR_2) \\
[\![p_i : DecJump(r_1, p_l)]\!]_2 &:= & i(p_i, R_1, R_2), s(PreR_1, R_1) &\Leftrightarrow i(p_{i+1}, PreR_1, R_2) \\
& & zero(R_1), i(p_i, R_1, R_2) &\Leftrightarrow i(p_l, R_1, R_2), zero(R_1) \\
[\![p_i : DecJump(r_2, p_l)]\!]_2 &:= & i(p_i, R_1, R_2), s(PreR_2, R_2) &\Leftrightarrow i(p_{i+1}, R_1, PreR_2) \\
& & zero(R_2), i(p_i, R_1, R_2) &\Leftrightarrow i(p_l, R_1, R_2), zero(R_2)
\end{aligned}
$$

Fig. 2.   2CM encoding in CHR

assuming that CT defines only the $=$ symbol which is interpreted as before and which, as before, cannot be used in the body of rules. Indeed, as we show in Figure 2, under these hypothesis we can encode 2CMs into CHR. The basic idea of this encoding is that to represent the values of the registers we use chains (conjunctions) of atomic formulas of the form $s(R_1, SuccR_1)$, $s(SuccR_1, SuccR_1')\ldots$ (recall that R_1, $SuccR_1$, $SuccR_1'$ are variables and we have countably many variables; moreover recall that the CHR computation mechanism avoids variables capture by using fresh names for variables each time a rule is used).

As we discuss in the conclusions this encoding, suggested by Jon Sneyers in a review of a previous version of this paper, is similar to those existing in the field of concurrency theory. Nevertheless, there are important technical differences. In particular, it is worth noting that for the correctness of the encoding it is essential that pattern matching rather than unification is used when applying rules: In fact this ensures that in the case of the decrement only one of the two instructions can match the goal and therefore can be used. The correctness of the encoding is stated by the following theorem whose proof is immediate.

THEOREM 3.3.   *A 2CM $M(0,0)$ halts with output $k > 0$ (or $k = 0$) if and only if the goal $zero(R_1) \wedge zero(R_2) \wedge i(1, R_1, R_2)$ in the program $[\![M(0,0)]\!]_2$ produces a qualified answer*

$$
\begin{aligned}
\exists_{-R_1, R_2} (\, & i(p_j, SuccR_1^k, R_2') \wedge zero(R_1) \wedge s(R_1, SuccR_1^1) \wedge \\
& \bigwedge_{i=1}^{k-1} s(SuccR_1^i, SuccR_1^{i+1}) \wedge H), \\
& \text{where } Fv(H) \cap \{R_1, SuccR_1^1, \ldots, SuccR_1^k\} = \emptyset
\end{aligned}
$$

(or $\exists_{-R_1, R_2} (i(p_j, R_1, R_2') \wedge zero(R_1) \wedge H)$, where $Fv(H) \cap \{R_1\} = \emptyset$).

Previous Theorems state a separation result between CHR and CHR_1, however this is rather weak since the real implementations of CHR usually consider a non-trivial constraint theory which includes function symbols. Therefore we are interested in proving finer separation results which hold for Turing powerful languages. This is the content of the following section.

4. SEPARATING CHR AND CHR$_1$

In this section we consider a generic constraint theory CT which allows the built-in predicate $=$ and we assume that the signature contains at least a constant and a function (of arity > 0) symbol. We have seen that in this case both CHR and CHR_1 are Turing powerful, which means that in principle one can always encode CHR into CHR_1. The question is how difficult and how acceptable such an encoding is and this question can have important practical consequences: for example, a distributed algorithm can be implemented in one language in a reasonably simple way and cannot be implemented in another (Turing powerful) language, unless one introduces rather complicated data structures or loses some compositionality properties (see [Vigliotti et al. 2007]).

We prove now that, when considering *acceptable encodings* and generic goals (whose components can share variables) CHR cannot be embedded into CHR_1 while preserving data sufficient answers. As a corollary we obtain that also qualified answers cannot be preserved. This general result is obtained by proving two more specific results.

First we have to formally define what an acceptable encoding is. We do this by giving a generic definition, which will be used also in the next section, which considers separately program and goal encodings. Hence in the following we denote by CHR_x some CHR (sub)language and assume that $\mathcal{P}_x$ is the set of all the CHR_x programs while $\mathcal{G}_x$ is the set of possible CHR_x goals. Usually the sub-language is defined by suitable syntactic restrictions, as in the case of CHR_1, however in some cases we will use also a semantic characterization, that is, by a slight abuse of notation, we will identify a sub-language with a set of programs having some specific semantic property. A *program encoding* of CHR_x into CHR_y is then defined as any function $[\![\ ]\!] : \mathcal{P}_x \to \mathcal{P}_y$. To simplify the treatment we assume that both the source and the target language of the program encoding use the same built-in constraints semantically described by a theory CT. Note that we do not impose any other restriction on the program translation (which, in particular, could also be non compositional).

Next we have to define how the initial goal of the source language has to be translated into the target language. Analogously to the case of programs, the goal encoding is a function $[\![\ ]\!]_g : \mathcal{G}_x \to \mathcal{G}_y$, however here we require that such a function is compositional (actually, an homomorphism) with respect to the conjunction of atoms, as mentioned in the introduction. Moreover, since both the source and the target language share the same constraint theory, we assume that the built-ins present in the goal are left unchanged. These assumptions essentially mean that our encoding respects the structure of the original goal and does not introduce new relations among the variables which appear in the goal. Note that we differentiate the goals $\mathcal{G}_x$ of the source language from those $\mathcal{G}_y$ of the target one because, in principle, a CHR_y program could use some user defined predicates which are not allowed in the goals of the original program – this means that the signatures of (language of) the original and the translated program could be different. Note also that the following definitions are parametric with respect to 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.

Finally, as mentioned before, we are interested in preserving data sufficient or qualified answers. Hence we have the following definition.

Definition 4.1 (*Acceptable encoding*). Let $\mathcal{G}$ be a class of CHR goals and let CHR_x and CHR_y be two CHR (sub)languages. An *acceptable encoding* of CHR_x into CHR_y, for the class of goals $\mathcal{G}$, is a pair of mappings $[\![\ ]\!] : \mathcal{P}_x \to \mathcal{P}_y$ and $[\![\ ]\!]_g : \mathcal{G}_x \to \mathcal{G}_y$ which satisfy the following conditions:

(1) $\mathcal{P}_x$ and $\mathcal{P}_y$ share the same CT;
(2) for any goal $(A, B) \in \mathcal{G}_x$, $[\![A, B]\!]_g = [\![A]\!]_g, [\![B]\!]_g$ holds. We also assume that the built-ins present in the goal are left unchanged;
(3) Data sufficient answers are preserved for the set of programs $\mathcal{P}_x$ and the class of goals $\mathcal{G}$, that is, for all $P \in \mathcal{P}_x$ and $G \in \mathcal{G}$, $\mathcal{SA}_P(G) = \mathcal{SA}_{[\![P]\!]}([\![G]\!]_g)$.

Moreover we define an *acceptable encoding for qualified answers* of CHR_x into CHR_y, for the class of goals $\mathcal{G}$, exactly as an acceptable encoding, with the exception that the third condition above is replaced by the following:

(3'). Qualified answers are preserved for the set of programs $\mathcal{P}_x$ and the class of goals $\mathcal{G}$, that is, for all $P \in \mathcal{P}_x$ and $G \in \mathcal{G}$, $\mathcal{QA}_P(G) = \mathcal{QA}_{[\![P]\!]}([\![G]\!]_g)$.

Obviously the notion of acceptable encoding for qualified answers is stronger than that one of acceptable encoding, since $\mathcal{SA}_P(G) \subseteq \mathcal{QA}_P(G)$ holds. Note also that, since we

consider goals as multisets, with the second condition in the above definition we are not requiring that the order of atoms in the goals is preserved by the translation: We are only requiring that the translation of A, B is the conjunction of the translation of A and of B, i.e. the encoding is homomorphic. Weakening this condition by 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 these languages. Moreover, homomorphic encoding are a quite common assumption in the papers studying expressiveness of concurrent languages, see for example [Palamidessi 2003].

We are now ready to prove our separation results. Next section considers only data sufficient answers.

4.1. Separating CHR and CHR$_1$ by considering data sufficient answers

In order to prove our first separation result we need the following lemma which states two key properties of CHR$_1$ computations. The first one says that if the conjunctive G, H with input constraint c produces a data sufficient answer d, then when considering one component, say G, with the input constraint d we obtain the same data sufficient answer. The second one states that when considering the subgoals G and H there exists at least one of them which allows to obtain the same data sufficient answer d also starting with an input constraint c' weaker than d.

LEMMA 4.2. *Let P be a CHR$_1$ program and let (c, G, H) be a goal, where c is a built-in constraint, G and H are multisets of CHR constraints. Let $V = Fv(c, G, H)$ and assume that (c, G, H) in P has the data sufficient answer d. Then the following holds:*

— *Both the goals (d, G) and (d, H) have the same data sufficient answer d.*
— *If $CT \models c \not\rightarrow d$ then there exists a built-in constraint c' such that $Fv(c') \subseteq V$, $CT \models c' \not\rightarrow d$ and at least one of the two goals (c', G) and (c', H) has the data sufficient answer d.*

PROOF. The proof of the first statement is straightforward (since we consider single headed programs). In fact, since the goal (c, G, H) has the data sufficient answer d in P, the goal (d, G) can either answer d or can produce a configuration where the user defined constraints are waiting for some guards to be satisfied in order to apply a rule r, but since the goal contains all the built-in constraints in the answer all the guards are satisfied letting the program to answer d.

We prove the second statement. Let

$$\delta = \langle (c, G, H), \emptyset, \emptyset \rangle \longrightarrow^* \langle \emptyset, \emptyset, d' \rangle \not\rightarrow$$

be the derivation producing the data sufficient answer $d = \exists_{-V} d'$ for the goal (c, G, H).

By definition of derivation and since by hypothesis $CT \models c \not\rightarrow d$, δ must be of the form

$$\langle (c, G, H), \emptyset, \emptyset \rangle \longrightarrow^* \langle (c_1, G_1), S_1, d_1 \rangle \longrightarrow \langle (c_2, G_2), S_2, d_2 \rangle \longrightarrow^* \langle \emptyset, \emptyset, d' \rangle \not\rightarrow,$$

where for $i \in [1, 2]$, c_i and d_i are built-in constraints such that $CT \models c_1 \wedge d_1 \not\rightarrow d$ and $CT \models c_2 \wedge d_2 \rightarrow d$. We choose $c' = \exists_{-V}(c_1 \wedge d_1)$. By definition of derivation and since P is a CHR$_1$ program, the transition

$$\langle (c_1, G_1), S_1, d_1 \rangle \longrightarrow \langle (c_2, G_2), S_2, d_2 \rangle$$

must be a rule application of a single headed rule r, which must match with a constraint k that was derived (in the obvious sense) by G or H. Without loss of generality, we can assume that k was derived from G. By construction c' suffices to satisfy the guards needed to reproduce k, which can then fire the rule r, after which all the rules needed to let the constraints of G disappear can fire. Therefore we have that

$$\langle (c', G), \emptyset, \emptyset \rangle \longrightarrow^* \langle \emptyset, \emptyset, d'' \rangle \not\rightarrow$$

where $CT \models \exists_{-V} d'' \leftrightarrow \exists_{-V} d' (\leftrightarrow d)$ and then the thesis follows. □

Note that Lemma 4.2 is not true anymore if we consider (multiple headed) CHR programs. Indeed if we consider the program P consisting of the single rule

$$\texttt{rule} @ H, H \Leftrightarrow \texttt{true} \mid c$$

then the goal (H, H) has the data sufficient answer c in P, but for each constraint c' the goal (H, c') has no data sufficient answer in P. With the help of the previous lemma we can now prove our main separation result. The idea of the proof is that any possible encoding of the rule

$$\texttt{r} @ H, G \Leftrightarrow \texttt{true} \mid c$$

into CHR_1 would either produce more answers for the goal H (or G), or would not be able to provide the answer c for the goal H, G. Using the notation introduced in Definition 2.8 and considering $\subseteq$ as multiset inclusion, we have then the following.

THEOREM 4.3.   *Let $\mathcal{G}$ be a class of goals such that if H is a head of a rule then $K \in \mathcal{G}$ for any $K \subseteq H$. Then, for $n \geq 2$, there exists no acceptable encoding of $\text{CHR}_{n,d}$ in CHR_1 for the class $\mathcal{G}$.*

PROOF.   The proof is by contradiction. Assume that there exists an acceptable encoding $[\![\ ]\!] : \mathcal{P}_{n,d} \to \mathcal{P}_1$ and $[\![\ ]\!]_g : \mathcal{G}_{n,d} \to \mathcal{G}_1$ of $\text{CHR}_{n,d}$ into CHR_1 for the class of goals $\mathcal{G}$ and let P be the program consisting of the single rule

$$\texttt{r} @ H, G \Leftrightarrow \texttt{true} \mid c.$$

Assume also, that c (restricted to the variables in H, G) is not the weakest constraint, i.e. assume that there exists d such that $CT \models d \not\rightarrow \exists_{-V} c$ where $V = Fv(H, G)$. Note that this assumption does not imply any loss of generality, since, as mentioned at the beginning of this section, we assume that the constraint theory allows the built-in predicate $=$ and the signature contains at least a constant and a function (of arity > 0) symbol.

Since the goal (H, G) has the data sufficient answer $\exists_{-V} c$ in the program P and since the encoding preserves data sufficient answers, the goal $[\![(H, G)]\!]_g$ has the data sufficient answer $\exists_{-V} c$ also in the program $[\![P]\!]$. From the compositionality of the translation of goals and the previous Lemma 4.2 it follows that there exists a constraint c' such that $Fv(c') \subseteq V$, $CT \models c' \not\rightarrow \exists_{-V} c$ and at least one of the two goals $[\![(c', H)]\!]_g$, and $[\![(c', G)]\!]_g$ has the data sufficient answer c in the encoded program $[\![P]\!]$.

However neither (c', H) nor (c', G) has any data sufficient answer in the original program P. This contradicts the fact that there exists an acceptable encoding of $\text{CHR}_{n,d}$ into CHR_1 for the class of goals $\mathcal{G}$, thus concluding the proof. □

Obviously, previous theorem implies that (under the same hypothesis) no acceptable encoding for qualified answers of $\text{CHR}_{n,d}$ into CHR_1 exists, since $\mathcal{SA}_P(G) \subseteq \mathcal{QA}_P(G)$. The hypothesis we made on the class of goals $\mathcal{G}$ is rather weak, as typically heads of rules have to be used as goals. From Theorem 4.3 we have immediately the following.

COROLLARY 4.4.   *Let $\mathcal{G}$ be a class of goals such that if H is a head of a rule then $K \in \mathcal{G}$ for any $K \subseteq H$. Then, for $n \geq 2$, there exists no acceptable encoding (for qualified answers) of CHR_n in CHR_1 for the class $\mathcal{G}$.*

As an example of the application of the previous theorem consider the program (from [Frühwirth 1998]) contained in Figure 3 which allows one to define the user-defined constraint *Lessequal* (to be interpreted as $\leq$) in terms of the only built-in constraint $=$ (to be interpreted as syntactic equality). For instance, given the goal $\{Lessequal(A, B), Lessequal(B, C), Lessequal(C, A)\}$ after a few computational steps the program will answer $A = B, B = C, C = A$. Now, for obtaining this behaviour, it is essential

reflexivity @ $Lessequal(X,Y) \Leftrightarrow X = Y \mid$ true
antisymmetry @ $Lessequal(X,Y), Lessequal(Y,X) \Leftrightarrow X = Y$
transitivity @ $Lessequal(X,Y), Lessequal(Y,Z) \Rightarrow Lessequal(X,Z)$

Fig. 3.   A program for defining $\leq$ in CHR

to use multiple heads, as already claimed in [Frühwirth 1998] and formally proved by the previous theorem. In fact, following the lines of the proof of Theorem 4.3, one can show that if a single headed program P' is any translation of the program in Figure 3 which produces the correct answer for the goal above, then there exists a subgoal which has an answer in P' but not in the original program.

4.2. Separating CHR and CHR$_1$ by considering qualified answers

Theorem 4.3 assumes that programs have non trivial data sufficient answers. Nevertheless, since qualified answers are the most interesting ones for CHR programs, one could wonder what happens when considering the CHR$_{n,t}$ language (see Definition 2.8).

Here we prove that also CHR$_{n,t}$ cannot be encoded into CHR$_1$. The proof of this result is somehow easier to obtain since the multiplicity of atomic formulae here is important. In fact, if $u(x,y)$ is a user-defined constraint, the meaning of $u(x,y), u(x,y)$ does not necessarily coincide with that one of $u(x,y)$. This is well known also in the case of logic programs (see any article on the S-semantics of logic programs): consider, for example, the program:

$$u(x,y) \Leftrightarrow x = a \qquad u(x,y) \Leftrightarrow y = b$$

which is essentially a pure logic program written with the CHR syntax. Notice that when considering an abstract operational semantics, as the one that we consider here, the presence of commit-choice does not affect the possible results. For example, in the previous program when reducing the goal $u(x,y)$ one can always choose (non deterministically) either the first or the second rule.

Now the goal $u(x,y), u(x,y)$ in such a program has the (data sufficient) answer $x = a, y = b$ while this is not the case for the goal $u(x,y)$ which has the answer $x = a$ and the answer $y = b$ (of course, using guards one can make more significant examples). Thus, when considering user-defined predicates, it is acceptable to distinguish $u(x,y), u(x,y)$ from $u(x,y)$, i.e. to take into account the multiplicity. This is not the case for "pure" built-in constraints, since the meaning of a (pure) built-in is defined by a first order theory CT in terms of logical consequences, and from this point of view $b \wedge b$ is equivalent to b.

In order to prove our result we need first the following result which states that, when considering single headed rules, if the goal is replicated then there exists a computation where at every step a rule is applied twice. Hence it is easy to observe that if the computation will terminate producing a qualified answer which contains an atomic user-defined constraint, then such a constraint is replicated. More precisely we have the following Lemma whose proof is immediate.

LEMMA 4.5.   *Let P be a CHR$_1$ program. If (G,G) is a goal whose evaluation in P produces a qualified answer (c,H) containing the atomic user-defined constraint k, then the goal (c,G,G) has a qualified answer containing (k,k).*

Hence we can prove the following separation result.

THEOREM 4.6.   *Let $\mathcal{G}$ be a class of goals such that if H is a head of a rule then $K \in \mathcal{G}$ for any $K \subseteq H$. Then, for $n \geq 2$, there exists no acceptable encoding for qualified answers of CHR$_{n,t}$ into CHR$_1$ for the class $\mathcal{G}$.*

PROOF. The proof will proceed by contradiction. Assume that there exists an acceptable encoding for qualified answers $[\![\ ]\!] : \mathcal{P}_{n,t} \to \mathcal{P}_1$ and $[\![\ ]\!]_g : \mathcal{G}_{n,t} \to \mathcal{G}_1$ of $\mathrm{CHR}_{n,t}$ into CHR_1 for the class of goals $\mathcal{G}$ and let P be the program consisting of the single rule:

$$\texttt{r} \ @ \ H, H \Leftrightarrow \texttt{true} \mid k$$

where k is an atomic user-defined constraint. The goal (H, H) in P has a qualified answer k (note that for each goal G, P has no trivial data sufficient answers different from G).

Therefore, by definition of acceptable encoding for qualified answers, the goal $[\![(H, H)]\!]_g$ in $[\![P]\!]$ has a qualified answer k (with the built-in constraint $\texttt{true}$). Since the compositionality hypothesis implies that $[\![(H, H)]\!]_g = [\![H]\!]_g, [\![H]\!]_g$, from Lemma 4.5 it follows that $[\![(H, H)]\!]_g$ in program $[\![P]\!]$ has also a qualified answer (k, k), but this answer cannot be obtained in the program with multiple heads thus contradicting one of the hypothesis on the acceptable encoding for qualified answers. Therefore such an encoding cannot exist. □

From previous theorem and Theorem 4.3 follows that, in general, no acceptable encoding for qualified answers of CHR in CHR_1 exists.

COROLLARY 4.7. *Let $\mathcal{G}$ be a class of goals such that if H is a head of a rule then $K \in \mathcal{G}$ for any $K \subseteq H$. Then there exists no acceptable encoding (for qualified answers) of CHR in CHR_1 for the class $\mathcal{G}$.*

5. A HIERARCHY OF LANGUAGES

After having shown that multiple heads increase the expressive power with respect to the case of single heads, it is natural to ask whether considering a different number of atoms in the heads makes any difference. In this section we show that this is indeed the case, since we prove that, for any $n > 1$, there exists no acceptable encoding (for qualified answers) of CHR_{n+1} into CHR_n. Thus, depending on the number of atoms in the heads, we obtain a chain of languages with increasing expressive power.

In order to obtain this generalization, we need to strengthen the requirement on acceptable encodings — only for data sufficient answers — given in Definition 4.1. More precisely, we now require that goals are unchanged in the translation process. This accounts for a "black box" use of the program: we do not impose any restriction on the program encoding, provided that the interface remains unchanged. Hence, in the following theorem we call "goal-preserving acceptable encoding" an acceptable encoding (according to Definition 4.1) where the function $[\![G]\!]_g$ which translates goals is the identity.

We have, then, the following result where we use the notation of Definition 2.8.

THEOREM 5.1. *Let $\mathcal{G}$ be the class of all possible goals. There exists no goal-preserving acceptable encoding of $\mathrm{CHR}_{n+1,d}$ in CHR_n for the class $\mathcal{G}$.*

PROOF. The proof will proceed by contradiction. Assume that there exists a goal-preserving acceptable encoding of $\mathrm{CHR}_{n+1,d}$ in CHR_n for the class $\mathcal{G}$ and let P be the following $\mathrm{CHR}_{n+1,d}$ program:

$$\texttt{rule} \ @ \ h_1 \ldots h_{n+1} \Leftrightarrow \texttt{true} \mid d$$

where $V = Fv(h_1 \ldots h_{n+1})$, d is a built-in constraint different from $\texttt{false}$ (i.e. $CT \models d \not\leftrightarrow \texttt{false}$ holds) such that $Fv(d) \subseteq V$. Hence given the goal $G = h_1 \ldots h_{n+1}$ the program P has the data sufficient answer d.

Observe that every goal with at most n user defined constraints has no data sufficient answer in P. Now consider a run of G in $[\![P]\!]$ (where $[\![P]\!]$ is the encoding of the program P) with final configuration $\langle \emptyset, \emptyset, d' \rangle$, where $CT \models \exists_{-V}(d') \leftrightarrow d$:

$$\delta = \langle G, \emptyset, \emptyset \rangle \to^* \langle H_i, G_i, d_i \rangle \to \langle H_{i+1}, G_{i+1}, d_{i+1} \rangle \to^* \langle \emptyset, \emptyset, d' \rangle \not\to,$$

where, without loss of generality, we can assume that in the derivation δ, for any configuration $\langle H', G', c' \rangle$ we can use either a Simplify or a Propagate transition only if H' does not contain built-ins and G_i is the last goal to be reduced in the run by using either a Simplify or a Propagate transition. Therefore G_i has at most n user-defined constraints, $H_i = \emptyset$ and let $r \in [\![P]\!]$ be the last rule used in δ (to reduce G_i). Since d is a built-in constraint, r can be only of the following form $H \Leftrightarrow C \mid C'$, where H has at most n user defined constraints. In this case $G_{i+1} = \emptyset$ and H_{i+1} contains only built-in predicates. Then

$$CT \models d_i \rightarrow \exists_{Fv(H)}((G_i = H) \wedge C) \text{ and}$$

$$CT \models (d_i \wedge C' \wedge (G_i = H)) \not\leftrightarrow \texttt{false}.$$

By construction the goal (G_i, d_i) has the data sufficient $\exists_{-Fv(G_i, d_i)}(d')$ in $[\![P]\!]$. But the goal (G_i, d_i) has no data sufficient answer in P thus contradicting one of the hypothesis on the goal-preserving acceptable encoding. Therefore such an encoding cannot exist. $\square$

Similarly to the development in the previous section, we now consider the case where the program has only qualified answers and no trivial data sufficient answers. Notice that in this case we do not require anymore that the translation of goals is the identity (we only require that it is compositional, as usual).

THEOREM 5.2.   *Let $\mathcal{G}$ be a class of goals such that if H is a head of a rule then $K \in \mathcal{G}$ for any $K \subseteq H$. There exists no acceptable encoding for qualified answers of $CHR_{n+1,t}$ in CHR_n for the class $\mathcal{G}$.*

PROOF.   The proof is by contradiction. Assume that there exists an acceptable encoding for qualified answers $[\![\ ]\!] : \mathcal{P}_{n+1,t} \rightarrow \mathcal{P}_n$ and $[\![\ ]\!]_g : \mathcal{G}_{n+1,t} \rightarrow \mathcal{G}_n$ of $CHR_{n+1,t}$ in CHR_n for the class of goals $\mathcal{G}$ and let P be the following $CHR_{n+1,t}$ program:

$$\texttt{rule} @ h_1 \ldots h_{n+1} \Leftrightarrow true \mid k$$

where $V = Fv(h_1 \ldots h_{n+1})$ and k is an atomic user defined constraint such that $Fv(k) \subseteq V$. Hence given the goal $G = h_1 \ldots h_{n+1}$ the program P has only the qualified answer k and since k is an atomic user defined constraint, we have that $k \neq (h_1 \ldots h_{n+1})$.

Observe that every goal with at most n user defined constraints has only itself as qualified answer in P. Then since the encoded program has to preserve all the qualified answers in the original P, every goal $[\![G_n]\!]_g$, where G_n has at most n user defined constraints, has a qualified answer G_n in $[\![P]\!]$.

Therefore, if we denote by $G_n = h_1 \ldots h_n$, by previous observation and by definition of qualified answers, we have that there exist two derivations

$$\langle [\![G_n]\!]_g, \emptyset, \emptyset \rangle \rightarrow^* \langle \emptyset, G'_n, d \rangle \not\rightarrow \text{ and } \langle [\![h_{n+1}]\!]_g, \emptyset, \emptyset \rangle \rightarrow^* \langle \emptyset, h'_{n+1}, d' \rangle \not\rightarrow,$$

such that

$$CT \models G_n \leftrightarrow \exists_{-Fv([\![G_n]\!]_g)}(G'_n \wedge d) \text{ and } CT \models h_{n+1} \leftrightarrow \exists_{-Fv([\![h_{n+1}]\!]_g)}(h'_{n+1} \wedge d').$$

Without loss of generality, we can assume that

$$Fv(G'_n, d) \cap Fv(h'_{n+1}, d') \subseteq Fv([\![G_n]\!]_g) \cap Fv([\![h_{n+1}]\!]_g).$$

Now consider the goal G, from what previously said we have that:

$$\langle [\![G]\!]_g, \emptyset, \emptyset \rangle \rightarrow^* \langle [\![h_{n+1}]\!]_g, G'_n, d \rangle$$

but we also know that $\langle [\![h_{n+1}]\!]_g, \emptyset, \emptyset \rangle \rightarrow^* \langle \emptyset, h'_{n+1}, d' \rangle \not\rightarrow$ and this cannot be prevented by any step in the previous run, thus we obtain:

$$\langle [\![G]\!]_g, \emptyset, \emptyset \rangle \rightarrow^* \langle \emptyset, (G'_n, h'_{n+1}), d \wedge d' \rangle,$$

where $CT \models G \leftrightarrow \exists_{-Fv(\llbracket G \rrbracket_g)}(G'_n \wedge h'_{n+1} \wedge d \wedge d')$. Since G is not a qualified answer for the goal G in P and since $\llbracket P \rrbracket$ is an acceptable encoding of P in CHR_n, we have that there exists $\{h'_{j_1}, \ldots h'_{j_s}\} \subseteq \{G'_n, h'_{n+1}\}$, with $s \leq n$, such that $\langle \emptyset, (h'_{j_1}, \ldots h'_{j_s}), d \wedge d' \rangle \rightarrow \langle G', H', d'' \rangle$ in $\llbracket P \rrbracket$. Then, since $CT \models G \leftrightarrow \exists_{-Fv(\llbracket G \rrbracket_g)}(G'_n \wedge h'_{n+1} \wedge d \wedge d')$, we have that there exists $\{h_{j_1}, \ldots h_{j_s}\} \subseteq G$ such that

$$CT \models h_{j_1}, \ldots h_{j_s} \leftrightarrow \exists_{-Fv(\llbracket h_{j_1}, \ldots h_{j_s} \rrbracket_g)}(h'_{j_1}, \ldots h'_{j_s} \wedge d \wedge d')$$

and therefore $h_{j_1}, \ldots h_{j_s}$ is not a qualified answer for $\llbracket h_{j_1}, \ldots h_{j_s} \rrbracket_g$ in $\llbracket P \rrbracket$ (since it is always possible to make another derivation step from $h_{j_1}, \ldots h_{j_s}$ in $\llbracket P \rrbracket$).

But, by previous observations, the same goal has itself as answer in P thus contradicting the fact that there exists an acceptable encoding for qualified answers of $\mathrm{CHR}_{n+1,t}$ in CHR_n. □

Notice that an immediate generalization of previous Theorem 5.2 implies that also under the weaker assumption of compositionality (rather than identity) for the translation of goals, no acceptable encoding for qualified answers for general CHR_{n+1} programs (including programs with data sufficient answers) into CHR_n exists. Therefore, from Theorem 5.2 we have immediately the following.

COROLLARY 5.3.   *Let $\mathcal{G}$ be a class of goals such that if H is a head of a rule then $K \in \mathcal{G}$ for any $K \subseteq H$. There exists no acceptable encoding for qualified answers of CHR_{n+1} in CHR_n for the class $\mathcal{G}$.*

It is also worth noticing that the previous results depends on the class of considered goals. In fact, if we limit the class of intended goals for a program and assume that some predicates in the translated program cannot be used in the goals, then one can easily encode a CHR_n program into a CHR_2 one. Consider for example the program consisting of the single rule

$$\mathtt{rule} @ h_0 \ldots h_n \Leftrightarrow C \mid B$$

and assume that the only valid goal for such a program is $h_0 \ldots h_n$, while $i_1, \ldots, i_n$ are fresh user-defined constraints that cannot be used in the goals. Then the following CHR_2 program is equivalent to the original one

$$\mathtt{r_1} @ h_0, h_1 \Leftrightarrow i_1$$
$$\mathtt{r_2} @ h_2, i_1 \Leftrightarrow i_2$$
$$\ldots$$
$$\mathtt{r_n} @ h_n, i_{n-1} \Leftrightarrow C \mid B$$

This restriction on fresh user-defined constraints to be used only in the encoding is rather strong, since all logic programming languages (including CHR) allow to use in the goals all the predicate names used in the program. In fact, essentially all the existing semantics for logic languages define the semantics of a program in a goal independent way, referring to all the possible predicates used in a program (or in the given signature). Nevertheless, from a pragmatic point of view it is meaningful to define a class of acceptable goals for a program and then to consider encoding, semantics etc, only w.r.t. that class of goals. In this respect it would be interesting to identify weaker conditions on goals and predicate names which allow to encode CHR_{n+1} into CHR_n (see also Section 6).

6. CONCLUSIONS AND RELATED WORKS

In this paper we have shown that multiple heads augment the expressive power of CHR. Indeed we have seen that the single head CHR language, denoted by CHR_1, is not Turing powerful when the underlying signature (for the constraint theory) does not contain function

symbols, while this is not the case for CHR. Moreover, by using a technique based on language encoding, we have shown that CHR is strictly more expressive than CHR_1 also when considering a generic constraint theory, under some reasonable assumptions (mainly, compositionality of the translation of goals). Finally we have shown that, under some slightly stronger assumptions, in general the number of atoms in the head of rules affects the expressive power of the language. In fact we have proved that CHR_n (the language containing at most n atoms in the heads of rules) cannot be encoded into CHR_m, with $n > m$.

There exists a very large literature on the expressiveness of concurrent languages, however there are only few papers which consider the expressive power of CHR. A recent one is [Sneyers 2008], where Sneyers 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. Moreover [Sneyers 2008] proves essentially the same result given in Theorem 3.3 by using Turing machines rather than Minsky machines. Both Theorems 3.2 and 3.3 were contained in the short version of this paper [Di Giusto et al. 2009], submitted before [Sneyers 2008] was published and both these results, including the encoding of the Minsky machine, were suggested by Jon Sneyers in the review of an older version ([Di Giusto et al. 2008]) of [Di Giusto et al. 2009]. It is worth noting that very similar encoding exists in the field of process algebras. For example, in [Busi et al. 2004] an encoding of Minsky machines in a dialect of CCS is provided which represents the value n of a register by using a corresponding number of parallel processes connected in a suitable way. This is similar to the idea exploited in Section 3, where we encoded the value n of a registers by using a conjunction (the CHR analogous of CCS parallel operator) of n atomic formulas.

Another related study is [Sneyers et al. 2005], 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 2001; 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 completely different from ours, even though it would be interesting to compare CHR and CHR_1 in terms of complexity.

When moving to other languages, somehow related to our paper is the work by Zavattaro [Zavattaro 1998] where the coordination languages Gamma [Banâtre and Métayer 1993] and Linda [Gelernter and Carriero 1992] are compared in terms of expressive power. Since Gamma allows multiset rewriting it reminds CHR multiple head rules, however the results of [Zavattaro 1998] are rather different from ours, since a process algebraic view of Gamma and Linda is considered where the actions of processes are atomic and do not contain variables. On the other hand, our results depend directly on the presence of logic variables in the CHR model of computation. Relevant for our approach is also [de Boer and Palamidessi 1994] which introduces the original approach to language comparison based on encoding, even though in this paper rather different languages with different properties are considered.

In [Laneve and Vitale 2008] Laneve and Vitale show that a language for modeling molecular biology, called κ-calculus, is more expressive than a restricted version of the calculus, called nano-κ, which is obtained by restricting to "binary reactants" only (that is, by allowing at most two process terms in the left hand side of rules, while n terms are allowed in κ). This result is obtained by showing that, under some specific assumptions, a particular (self-assembling) protocol cannot be expressed in nano-κ, thus following a general technique which allows to obtain separation results by showing that (under some specific hypothesis) a problem can be solved in a language and not in another one (see also [Palamidessi 2003] and [Vigliotti et al. 2007]). This technique is rather different from the one we used, moreover also the assumptions on the translation used in [Laneve and Vitale 2008] are different

from ours. Nevertheless, since κ (and nano-κ) can be easily translated in CHR, it would be interesting to see whether some results can be exported from a language to another. We left this as future work.

We also plan to investigate what happens when considering the translation of CHR since many CHR implementations are built on top of a Prolog system, by using a compiler which translates CHR programs to Prolog. Our technical lemmata about CHR_1 can be adapted to what is called [Apt 1996] "pure Prolog", that is, a logic programming language which uses the leftmost selection rule and the depth-first search. Hence it is easy to show that, under our assumptions, CHR cannot be encoded in pure Prolog. However, implemented "real" Prolog systems are extensions of pure Prolog obtained by considering specific built-ins for arithmetic and control, and when considering these built-ins some of the properties we have used do not hold anymore (for example, this is the case of Lemma 4.2). Hence it would be interesting to see under which conditions CHR can be encoded in real Prolog systems, that is, which features of real Prolog (which are not present in pure Prolog) are needed to obtain an acceptable encoding of CHR. Finally we plan to extend our results to consider specific constraint theories (e.g. with only monadic predicates) and also taking into account the refined semantics defined in [Duck et al. 2004]. This latter semantics requires further work, because it allows an improved control on computations and some properties that we used do not hold anymore in this case.

ACKNOWLEDGMENT

References

ABDENNADHER, S. 1997. Operational semantics and confluence of constraint propagation rules. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP'97)*. 252–266.

APT, K. R. 1996. *From logic programming to Prolog*. Prentice-Hall, Inc.

BANÂTRE, J.-P. AND MÉTAYER, D. L. 1993. Programming by multiset transformation. *Commun. ACM 36,* 1, 98–111.

BUSI, N., GABBRIELLI, M., AND ZAVATTARO, G. 2004. Comparing recursion, replication, and iteration in process calculi. In *Thirtyfirst International Colloquium on Automata, Languages and Programming (ICALP'04)*. Lecture Notes in Computer Science Series, vol. 3142. Springer-Verlag, 307–319.

DE BOER, F. S. AND PALAMIDESSI, C. 1994. Embedding as a tool for language comparison. *Information and Computation 108,* 1, 128–157.

DI GIUSTO, C., GABBRIELLI, M., AND MEO, M. C. 2009. Expressiveness of multiple heads in CHR. In *SOFSEM 2009*. Lecture Notes in Computer Science Series, vol. 5404. Springer, 205–216.

DI GIUSTO, C., GABBRIELLI, M., AND MEO, M. C. April 2008. Expressiveness of multiple heads in CHR. *CoRR abs/0804.3351*.

DUCK, G. J., STUCKEY, P. J., DE LA BANDA, M. J. G., AND HOLZBAUR, C. 2004. The refined operational semantics of constraint handling rules. In *Proceedings of the 20th International Conference on Logic Programming (ICLP 2004)*. Lecture Notes in Computer Science Series, vol. 3132. Springer, 90–104.

FRÜHWIRTH, T. 1991. Introducing simplification rules. Tech. rep.

FRÜHWIRTH, T. 2002. As time goes by: Automatic complexity analysis of simplification rules. In *8th International Conference on Principles of Knowledge Representation and Reasoning, Toulouse, France*.

FRÜHWIRTH, T. W. 1998. Theory and practice of constraint handling rules. *J. Log. Program. 37,* 1-3, 95–138.

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

GELERNTER, D. AND CARRIERO, N. 1992. Coordination languages and their significance. *Commun. ACM 35,* 2, 96.

LANEVE, C. AND VITALE, A. 2008. Expressivity in the kappa family. *Electr. Notes Theor. Comput. Sci. 218*, 97–109.

MINSKY, M. 1967. *Computation: finite and infinite machines*. Prentice Hall.

PALAMIDESSI, C. 2003. Comparing the expressive power of the synchronous and asynchronous *pi*-calculi. *Mathematical. Structures in Comp. Sci. 13,* 5, 685–719.

SHAPIRO, E. Y. 1989. The family of concurrent logic programming languages. *ACM Comput. Surv. 21,* 3, 413–510.

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

SNEYERS, J., SCHRIJVERS, T., AND DEMOEN, B. 2005. The computational power and complexity of Constraint Handling Rules. In *Proceedings of the 2nd Workshop on Constraint Handling Rules (CHR'05)*, T. Schrijvers and T. Frühwirth, Eds. Number CW 421 in Dept. Computer Science, Technical report. Sitges, Spain, 3–17.

VAANDRAGER, F. W. 1993. Expressive results for process algebras. In *Proceedings of the REX Workshop on Sematics: Foundations and Applications*. Springer-Verlag, London, UK, 609–638.

VIGLIOTTI, M. G., PHILLIPS, I., AND PALAMIDESSI, C. 2007. Tutorial on separation results in process calculi via leader election problems. *Theor. Comput. Sci. 388,* 1-3, 267–289.

ZAVATTARO, G. 1998. On the incomparability of gamma and linda. Tech. rep., Amsterdam, The Netherlands.