A new approach to termination analysis of CHR

Dean Voets*, Paolo Pilozzi**, and Danny De Schreye

Department of Computer Science, K.U.Leuven, Belgium
Celestijnenlaan 200A, 3001 Heverlee
tel:+32 16 327641    fax: +32 16 327996
{Dean.Voets@cs, Paolo.Pilozzi@cs, Danny.DeSchreye@cs }.kuleuven.be

Abstract. We present a new approach to termination analysis of Constraint Handling Rules (CHR). Unlike current approaches, our approach has no restrictions on the kind of rules in the CHR program. We propose a termination condition that verifies conditions imposed on the dynamic process of adding constraints to the store, instead of a termination argument based on the comparison of sizes of consecutive computation states. We demonstrate the condition’s applicability on a set of terminating CHR programs, using a prototype analyzer. This analyzer is the first in-language automated termination analyzer for CHR programs.

Keywords: Constraint Handling Rules, Termination Analysis

1 Introduction

Constraint Handling Rules (CHR), created by Thom Frühwirth [4], is a relatively young member of the declarative programming languages family. It is a concurrent, committed-choice, logic programming language. CHR is constraint-based and has guarded rules that rewrite multisets of atomic formulas until they are solved. CHR defines three kinds of rules: simplification rules remove constraints and add new ones, propagation rules only add new constraints and simpagation rules are a combination of both. CHR’s simple syntax and semantics make it well-suited for implementing custom constraint solvers [4, 11, 12]. Particularly the latter feature of the language accounts for its success and impact on the research community.

Due to its multi-headed rules, CHR provides extra declarative expressivity compared to the single-headed rules of Logic Programming (LP). But at the same time, the operational behavior of multi-headed rules is more complex and harder to predict than that of single-headed ones. As a consequence, the importance of automated analysis tools, that assist a CHR-programmer to predict runtime properties of his/her programs is high. One of these properties is the termination of such programs.

* Supported by the Fund for Scientific Research - Flanders (Belgium) (F.W.O.): "Termination analysis for Constraint Handling Rules" FWO-project G0561-08
** Partly supported by the Fund for Scientific Research - Flanders (Belgium) (F.W.O.): "Termination analysis, crossing paradigm borders”. FWO-project G0320-06. Partly supported by IWT Flanders.
Although the language is strongly related to LP, termination analysis of CHR programs has received little attention. Termination of CHR without propagation has been addressed before in [5] and [10]. The approach of [5] is based on adapting well-known LP termination techniques directly to CHR. The approach of [10] is based on a transformation of CHR into Prolog.

Because propagation rules only add constraints, a fire-once policy is used for these rules. A condition based on this fire-once policy is needed to prove termination for such programs. In this paper we present a condition based on an interpretation of the constraints, such that propagation rules only add 'smaller' constraints and for other rules, the number of the 'largest' constraints decreases. Because of the fire-once policy, this condition implies termination. We implemented the method in a prototype analyzer and performed an experimental evaluation. The results were very satisfactory. Since the approach presented in [5] was not implemented, this analyzer is the first in-language automated termination analyzer for CHR.

The paper is organized as follows. In the next section, we introduce the basic aspects of CHR and adapt some concepts from termination analysis of LP to the CHR context. In Section 3, we introduce a termination condition for general CHR programs that is sufficient for proving termination. Next, we further refine the condition so that it can be automated. In Section 4, we discuss our implementation and experimental evaluation. Finally, Section 5 concludes the paper.

2 Preliminaries

2.1 Constraint Handling Rules

Syntax. A constraint in CHR is a first-order predicate. We distinguish between built-in constraints, predefined and solved by the underlying constraint theory, CT, and CHR constraints, user-defined and solved by a CHR program P. A CHR program is a finite set of CHR rules. Rules are of the form:

\[
\begin{align*}
\text{Simplification rule:} & \quad \text{Propagation rule:} \quad \text{Simpagation rule:} \\
true \land H & \iff [G \mid B]. \quad H \land true & \iff [G \mid B]. \quad H_1 \land H_2 & \iff [G \mid B]. \\
or H & \iff [G \mid B]. \quad or H & \iff [G \mid B].
\end{align*}
\]

The head \( H \) or \( H_1 \land H_2 \) is a conjunction of CHR constraints. The optional guard \( G \) is a conjunction of built-in constraints. The body \( B \) is a conjunction of built-in and CHR constraints. Empty conjuncts are denoted by the built-in constraint \( true \). As in Prolog syntax, conjuncts are separated by commas. Note that all rules can be written as simpagation rules.

Example 1 (Fibonacci). The CHR program below implements a Fibonacci algorithm. Natural numbers are written in the symbolic notation, using 0 and the successor functor \( s \). In this program, add/3 is a built-in constraint which defines addition on natural numbers written in symbolic notation.

\[
\text{fib}(N, M_1), \text{fib}(N, M_2) \iff M_1 = M_2, \text{fib}(N, M_1).
\]
\[ \text{fib}(0, M) \Rightarrow M = s(0). \]
\[ \text{fib}(s(0), M) \Rightarrow M = s(0). \]
\[ \text{fib}(s(s(N)), M) \Rightarrow \text{fib}(s(N), M1), \text{fib}(N, M2), \text{add}(M1, M2, M). \]

The first rule is a simplification rule that removes doubles. The other rules are propagation rules. Base cases are solved by the second and third rule. The last rule adds CHR constraints representing Fibonacci numbers and a built-in constraint relating their arguments.

**Operational Semantics.** A CHR program defines a state transition system, where a state is defined as a conjunction of CHR and built-in constraints, called the constraint store. The initial state or query is an arbitrary conjunction of constraints. Each state where either the built-in constraints are inconsistent (failed state), or no more transitions are possible, is called a final state.

**Definition 1 (Transition relation).** The transition relation, \( \rightarrow \), between states, given a constraint theory CT and a CHR program P, is defined as:

\[
H_1' \land H_2' \land D \rightarrow (B \land H_1' \land D) \theta \theta'
\]

if \( H_1 \setminus H_2 \Leftrightarrow G \mid B \) in P and \( \theta, \theta' \) are substitutions such that \( CT \models D \rightarrow (((H_1 \theta \land H_2 \theta) = (H_1' \land H_2')) \land G \theta \theta') \)

A CHR rule, \( H_1 \setminus H_2 \Leftrightarrow G \mid B \), is applicable to a conjunction of CHR constraints, \( H' = H_1' \land H_2' \), if \( H_1' \) matches head \( H_1 \) and \( H_2' \) matches head \( H_2 \) with matching substitution \( \theta \), such that the guard \( G \) evaluates to true, with answer substitution \( \theta' \), given the built-ins in the constraint store. The body \( B \) is a conjunction of built-in and CHR constraints, \( B \theta \theta' \) is added to the constraint store. Rule application is non-deterministic and committed-choice.

For more information about the operational semantics of CHR, we refer to [4].

A CHR program \( P \) with query \( I \) terminates, if all computations for \( P \) with query \( I \) end in a final state. Because propagation rules do not remove constraints from the store, a fire-once policy is used to prevent trivial non-termination. This fire-once policy ensures that no propagation rule fires more than once on the same multi-set of constraints. The propagation history implements this policy. For a more detailed description of the propagation history, we refer to [1].

CHR has no fairness guarantees: constraints in the body of a rule might never be selected in a computation. This means that our technique will not be able to prove termination if termination depends on the evaluation of built-in constraints from the body.

In the following example, we discuss a computation for the Fibonacci program discussed earlier.

**Example 2 (Fibonacci continued).** With a typical query \( \text{fib}(s(s(s(0)))), N) \), the last propagation rule adds two new \( \text{fib}/2 \) constraints to the store with lower first arguments. However, it does not remove the constraint that has fired the rule from the store. This propagation rule fires again on the added constraint \( \text{fib}(s(s(0)), N) \), adding again two CHR constraints with a lower first argument
to the store. The other two propagation rules resolve base cases, while the simplification rule removes duplicates. The constraint store:

\[ \text{fib}(s(s(0))), s(s(0))) \land \text{fib}(s(0)), s(0)) \land \text{fib}(0, s(0)) \]

is the final state. Without the simplification rule the answer would contain an additional \( \text{fib}(s(0), s(0)) \) constraint. □

Observe that, given any reasonable way to measure the size of the constraint store, the size of the store increases for this program. This means that standard approaches for proving termination of LP, which prove decreases in size between consecutive computation states, are not easily and immediately applicable to CHR. One would need to explicitly add an encoding of the propagation history to the representation of the store to make this work. In very recent work, some of the authors of this paper have worked out the details of such an approach. In this paper, we have chosen for a different type of termination condition for programs with propagation rules.

2.2 Termination Analysis

Termination analysis of logic programs has received a lot of attention (see for example [2]). In LP, consecutive computation states are compared using a norm and level mapping. A norm is a function which maps terms to natural numbers. A level mapping is a function which maps atoms to natural numbers. In this subsection we redefine some well-known concepts from LP to the CHR context.

We define \( \text{Term}_P \) as the set of all terms constructible from a program \( P \) and \( \text{Con}_P \) as the set of all constraints constructible from \( P \), with arguments from \( \text{Term}_P \).

Definition 2 (norm, level mapping). A norm is a mapping \( ||\cdot|| : \text{Term}_P \rightarrow \mathbb{N} \). A level mapping is a mapping \( |\cdot| : \text{Con}_P \rightarrow \mathbb{N} \).

We refer to \( |C| \) as the level value of \( C \). Several examples of norms and level mappings can be found in the literature on LP termination analysis [2]. Two well-known norms are list-length and term-size.

Definition 3 (list-length, term-size).

\( \text{List-length is defined as:} \)

\[ ||t_1|t_2|| = 1 + ||t_2|| \]

\[ ||t|||_1 = 0 \text{ otherwise.} \]

\( \text{with } t_1, t_2 \text{ and } t \text{ any term.} \]

\( \text{Term-size is defined as:} \)

\[ ||f(t_1, t_2, ..., t_n)||_t = 1 + \sum_{1 \leq i \leq n} ||t_i||_t \]

\[ ||t||_t = 0 \text{ otherwise} \]

\( \text{with } t_1, ..., t_n \text{ and } t \text{ any term.} \)

The most common kind of level mapping is the linear level mapping.

Definition 4 (linear level mapping). A linear level mapping is any level mapping which can be defined as:

\[ |\text{con}(t_1, ..., t_n)| = \text{con}_0 + \sum_{1 \leq i \leq n} \text{con}_i|t_i|, \]

\( \text{with } \text{con}_i \in \mathbb{N} \) only depending on the constraint symbol \( \text{con} \) and \( ||\cdot|| \) a norm.

We also adapt the notion of rigidity to the CHR context.
Definition 5 (Rigidity). A CHR constraint C is **rigid** w.r.t. a level mapping \(|.|\) iff \(\forall\) substitutions \(\theta\) : \(|C| = |C\theta|\). 

To prove termination, we need some information about the CHR constraints which can be added to the store during an execution of a program for a query. For this purpose, we define the call set.

Definition 6 (Call Set). Given a program \(P\) and a query \(I\), the call set for \(P\) with query \(I\), Call\((P, I)\), is the union, over all possible computations of \(P\) for \(I\), of all CHR-constraints which are added to the constraint store during that computation.

Usually, Call\((P, I)\) is specified using an abstraction. In what follows we describe the call set using types.

### 3 A new termination condition for CHR

In [5], concepts and ideas from LP termination analysis are adapted to CHR with simplification only. Termination is proved by showing a decrease between the removed and the added constraints, for each CHR rule in the program.

The extension to programs with propagation rules gives a totally new termination problem. In LP or CHR without propagation, each rule removes one predicate, for LP, and at least one constraint, for CHR without propagation. Termination is proven by measuring a decrease of the goal, or of the constraint store, for each rule application. For CHR with propagation, this approach seems infeasible because new constraints are added and no existing constraints are removed. As mentioned before, one would need to keep track of information regarding the propagation history to observe a decrease. Instead of a termination argument based on a comparison of sizes of consecutive computation states, we formulate and verify conditions imposed on the dynamic process of adding constraints to the store. We formulate conditions which guarantee that the entire computation only adds a finite number of constraints to the store. Due to the use of a propagation history, this implies termination.

First we prove that if only a finite number of CHR constraints are added to the constraint store, program \(P\) with query \(I\) terminates. Note that, if the same constraint is added multiple times to the constraint store, then we consider these additions as different.

Lemma 1 (Termination of a CHR program). A CHR program \(P\) with query \(I\) terminates iff there are a finite number of additions of CHR constraints to the constraint store during any execution of \(P\) for \(I\). 

Proof. \(\implies\): If \(P\) terminates for \(I\), then for any execution of \(P\) for \(I\), only a finite number of rules are applied. Therefore, only a finite number of CHR constraints are added to the store.

\(\impliedby\): Suppose there are only a finite number of CHR constraints added to the store during any execution of program \(P\) for \(I\). Each propagation rule can only
be fired a finite number of times because of the propagation history. Each simplification or simpagation rule removes at least one CHR constraint from the store. Therefore, a simpagation or simplification rule can only be applied a finite number of times. Since every rule can only be applied a finite number of times, $P$ terminates for $I$. □

3.1 Ranking Condition (RC) for CHR with substitutions

The next definition gives the first version of our Ranking Condition (RC). It is applicable to general CHR programs. Informally, a program satisfies the RC if each propagation rule only adds constraints which are smaller than the constraints in the head and each simplification or simpagation rule reduces the number of largest constraints in the rule.

**Definition 7 (Ranking condition for CHR with substitutions).** A program $P$ and a query $I$ satisfy the RC for CHR, w.r.t. level mapping $|.|$ iff every CHR constraint in $Call(P, I)$ is rigid w.r.t. $|.|$ and for each rule in $P$ and for every matching substitution $\theta$ and answer substitution $\theta'$ from Definition 1:

1. For a simplification or simpagation rule $H \setminus H_1, ..., H_n \Leftrightarrow G \mid B_1, ..., B_m$, with $n > 0$ and body-CHR constraints $B_k, ..., B_m$,
   let $p = \max \{|H_1\theta|, ..., |H_n\theta|, |B_k\theta\theta'|, ..., |B_m\theta\theta'|\}$. Then,
   the number of CHR constraints with level value $p$ is higher in $\{H_1\theta, ..., H_n\theta\}$ than in $\{B_k\theta\theta', ..., B_m\theta\theta'\}$

2. For a propagation rule: $H_1, ..., H_n \Rightarrow G \mid B_1, ..., B_m$, with body-CHR constraints $B_k, ..., B_m$:
   for all $i = 1, ..., n$ and $j = k, ..., m$: $|H_i\theta| > |B_j\theta\theta'|$. □

Our condition on simplification and simpagation rules in Definition 7 is more strict than the corresponding ranking condition for such rules in [5]. The reason for this is that in the presence of propagation rules, a multiset order decrease, as in [5], is insufficient to guarantee termination.

**Example 3 (Counterexample multiset order decrease).** Consider the program:

$$a(s(N)), a(N) \Leftrightarrow a(s(N)). \quad a(s(N)) \Rightarrow a(N).$$

The program does not terminate for any query $a(s(n))$, with $n$ any term. Our RC can not be fulfilled for the simplification rule. However, using a level mapping $|a(t)| = ||t||_t$ and a multiset order on conjunctions of constraints, as in [5], there is a decrease from head to body constraints for this rule. So a straightforward extension of the ranking condition of [5] is incorrect. □

**Theorem 1 (Sufficiency of the RC with substitutions).** Let program $P$ with query $I$ satisfy the RC with substitutions w.r.t. $|.|$, then all computations for $P$ with query $I$ terminate. □
Proof. In order to prove termination of a CHR program $P$ with a query $I$, Lemma 1 shows that it is sufficient to prove that the total number of CHR constraints, added during an execution of $P$ for $I$, is finite. As a base of the induction, we show that there is a maximal level value for the CHR constraints in the store and that only a finite number of CHR constraints with that level mapping can be added to the store. Using induction we prove that for each level value only a finite number of CHR constraints can enter the store.

Before we present the induction proof, first note that because of the rigidity of $\text{Call}(P, I)$ under $\mid \cdot \mid$, the level value of a constraint that matches the head of a rule cannot change anymore due to instantiations caused by answer substitutions of guards or built-in body constraints. So, level values of constraints in the store are static: the level values cannot change over time.

**Base case.** Let $max$ be the maximal level value of the constraints in $I$. Propagation rules only add constraints with a lower level value than $max$. Because of the RC, simplification or simpagation rules only add constraints with a level value smaller than or equal to the largest level value of the constraints matching the head of the rule. Every time a CHR constraint of level value $max$ is added by such a rule, the number of CHR constraints with level value $max$ decreases. So only a finite number $a_{max}$ of CHR constraints with level value $max$ are added to the store during any execution of $P$ for $I$.

**Induction step.** Let $a_{max}, \ldots, a_{n+1}$ be upper limits for the number of CHR constraints with level value $max, \ldots, n + 1$, w.r.t. $\mid \cdot \mid$, which can be added to the constraint store during an execution of $P$ with query $I$.

- $I$ only contains a finite number of constraints with level value $n$.
- If an instance of a propagation rule adds a CHR constraint with level value $n$ to the store, the CHR constraints matching the head all have a level value larger than $n$. Because of the upper limits $a_{max}, \ldots, a_{n+1}$ and the propagation history, every propagation rule can only add a finite number of CHR constraints with level value $n$.
- For every instance of a simplification or simpagation rule which adds a CHR constraint with level value $n$ to the store, there exists an $i : n \leq i \leq max$, such that the number of CHR constraints of level value $i$ decreases, and no constraint with a level value higher than $i$ is added to the store by this simplification or simpagation rule. This implies that only a finite number of constraints with level mapping $n$ can enter the store by simpagation or simplification rules, because after enough rule executions, there are no CHR constraints with a level value $n$ or higher left.

By induction, this proves that only a finite number of CHR constraints are added to the store. Therefore, Lemma 1 proves termination. □

**Example 4 (Fibonacci continued).** We prove termination for the Fibonacci example with the RC with substitutions.

Let $\text{fib}(n, m)$ be any query, with $n$ a ground term, representing a natural number in successor-notation and $m$ a free variable. One can infer that the
call set is the set \( \{ \text{fib}(n_1, m), \text{fib}(n_2, n_3) \mid n_1, n_2, n_3 \text{ ground terms, representing natural numbers and } m \text{ a free variable} \} \). As a norm, we use term-size. The level mapping is defined as \( |\text{fib}(n, m)|_t = |n|_t \). Clearly, the call set is rigid w.r.t. \(|.|_t\).

For the first rule, we have that for every matching substitution \( \theta \), the first term in every \( \text{fib}/2 \) constraint is substituted by the same ground term. The answer substitutions \( \theta' \) are the identity substitutions because this rule has no guard:

\[
|\text{fib}(N, M1)\theta| = |\text{fib}(N, M2)\theta| = |\text{fib}(N, M1)\theta\theta'| = |N\theta\theta'|_t
\]

All constraints have the same level value. There are two constraints in the head and one in the body, so this rule satisfies the RC with substitutions.

For the fourth rule we have that for every matching substitution \( \theta \), the term matching \( s(s(N)) \) is a ground term. The answer substitutions \( \theta' \) are empty, because the rule has no guard. For all matching and answer substitutions:

\[
|\text{fib}(s(s(N)), M)\theta| = 2 + |N\theta\theta'|_t > |\text{fib}(s(s(N)), M1)\theta\theta'| = 1 + |N\theta\theta'|_t \\
|\text{fib}(s(s(N)), M)\theta| = 2 + |N\theta\theta'|_t > |\text{fib}(N, M2)\theta\theta'| = |N\theta\theta'|_t
\]

The RC with substitutions is therefore satisfied, which implies termination for the considered queries.

\[\square\]

3.2 Ranking Condition for CHR

A disadvantage of the ranking condition of Definition 7, is that one has to consider all matching and answer substitutions. This cannot be done automatically because in general there can be infinitely many of such substitutions. In this section, we present a RC which is more suitable for automation. In order to estimate the effects of the matching substitutions, abstract norms and abstract level mappings are adapted from [3] and [9]. These functions map a variable to itself, instead of to zero. In order to estimate the effects of the answer substitutions, interargument relations are used.

Let \( N[\text{Var}_P] \) denote the set of polynomials over \( \text{Var}_P \), with natural coefficients.

**Definition 8 (abstract norm, abstract level mapping).** An abstract norm is a mapping \(|.| : \text{Term}_P \to N[\text{Var}_P] \), which is the identity function on \( \text{Var}_P \).

An abstract level mapping is a mapping \(|.| : \text{Con}_P \to N[\text{Var}_P] \).

Abstract list-length, abstract term-size and abstract linear level mappings are the obvious adaptations of Definitions 3 and Definition 4.

**Example 5 (Abstract linear level mappings).** Let \( L, \text{List}, N \) and \( M \) be variables. Typical examples of abstract linear level mappings are:

\[
|\text{mergesort}(L, \text{List})|_\alpha = 0 + 1.|L[\text{List}]|_\alpha = 1 + \text{List} \\
|\text{fib}(s(s(N)), M)|_\alpha = 0 + 1.|s(s(N))|_\alpha + 0.|M|_\alpha = 2 + N
\]

In general, an abstract level mapping maps constraints to arbitrary polynomials over \( N \). In order to compare the level values of the constraints w.r.t. an abstract level mapping, we define an ordering on polynomials over \( N \) [9].
Definition 9 (orderings on $\mathbb{N}[\text{Var}_P]$). Let $p$ and $q$ be two polynomials with $n$ variables. The quasi-ordering $\succeq$ is defined as $p \succeq q$ iff $p(x_1, \ldots, x_n) \geq q(x_1, \ldots, x_n)$ for all $x_1, \ldots, x_n \in \mathbb{N}$. The strict ordering $\succ$ is defined as $p \succ q$ iff $p(x_1, \ldots, x_n) > q(x_1, \ldots, x_n)$ for all $x_1, \ldots, x_n \in \mathbb{N}$. The equality between polynomials is defined as $p \approx q$ iff $p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n)$ for all $x_1, \ldots, x_n \in \mathbb{N}$. \hfill $\square$

The next example shows the orderings between three polynomials.

Example 6 (polynomial ordering).
Let $p(X, Y) = 1 + XY + 2X$, $q(X) = 2X$ and $z(X, Y) = XY + X$:
- $p(X, Y) \succ q(X)$, $p(X, Y) \succ z(X, Y)$.
- Neither $q(X) \succeq z(X, Y)$ nor $z(X, Y) \succeq q(X)$.

As stated, interargument relations are used to estimate the effect of the answer substitutions from Definition 1.

Definition 10 (Interargument relation). Let $P$ be a program and $p/n$ a built-in constraint in $P$. An interargument relation for $p/n$ is a relation $R_{p/n} \in \mathbb{N}^n$. $R_{p/n}$ is a valid interargument relation for $p/n$ w.r.t. a norm $|||., |||$, iff
\[ \forall t_1, \ldots, t_n \in \text{Term}_P : CT || p(t_1, \ldots, t_n) \implies (||t_1||, \ldots, ||t_n||) \in R_{p/n}. \]

Of course, the interargument relations are dependent on the chosen host-language. The following examples are for swi-prolog.

Example 7 (Interargument relations).

<table>
<thead>
<tr>
<th>built-in constraint</th>
<th>norm and interargument relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete(N1,N2,N3)</td>
<td>list-length: ${(n_1, n_2, n_3) \in \mathbb{N}^3</td>
</tr>
<tr>
<td>append(N1,N2,N3)</td>
<td>list-length: ${(n_1, n_2, n_3) \in \mathbb{N}^3</td>
</tr>
<tr>
<td>leq(N1,N2)</td>
<td>term-size: ${(n_1, n_2) \in \mathbb{N}^2</td>
</tr>
</tbody>
</table>

We define rigidity for abstract level mappings.

Definition 11 (Rigidity). A CHR constraint $C$ is rigid w.r.t. an abstract level mapping $|.|^\alpha$ iff $\forall$ substitutions $\theta : |C|^\alpha \approx |C\theta|^\alpha$. \hfill $\square$

For an instance of a simplification or simpagation rule, the maximal level value is used in our RC with substitutions from Definition 7. Because the maximum is not defined for a set of polynomials, we introduce a $\succeq$-maximal subset of constraints.

Definition 12 ($\succeq$-maximal subset of constraints). Let $C$ be a multiset of CHR constraints and $|.|^\alpha$ an abstract level mapping. $D$ is a $\succeq$-maximal subset of $C$ w.r.t. $|.|^\alpha$ iff $D$ is a non-empty multi-subset of $C$ such that:

- The constraints in $D$ have the same level value:
  \[ \forall \text{CHR-constraints } C_1, C_2 \in D : |C_1|^\alpha \approx |C_2|^\alpha \]
- There are no $|.|^\alpha$-larger constraints in $C \setminus D$:
  \[ \lnot (\exists C_1 \in D, \exists C_2 \in C \setminus D : |C_2|^\alpha \succeq |C_1|^\alpha) \]
Note that a multiset $C$ may have several $\geq$-maximal subsets.

We now reformulate our ranking condition from Definition 7, using abstract level mappings and interargument relations.

**Definition 13 (Ranking condition for CHR).**
A program $P$ and a query $I$ satisfy the RC for CHR, w.r.t. an abstract level mapping $|.|^\alpha$ and a set of interargument relations iff every constraint in $Call(P, I)$ is rigid w.r.t. $|.|^\alpha$ and for each rule in $P$ and for each substitution $\sigma$ such that the built-in constraints in the guard all satisfy their associated interargument relations:

1. For a simplification or simpagation rule $H \setminus H_1, \ldots, H_n \Rightarrow G \mid B_1, \ldots, B_m$, with $n > 0$, body-CHR constraints $B_k, \ldots, B_m$ and $D_1, \ldots, D_c$ all $\geq$-maximal subsets of $\{H_1, \ldots, H_n, B_k\sigma, \ldots, B_m\sigma\}$, then
   \[
   \forall i \in \{1, \ldots, c\} : \text{let } D_i = \{H_{i1}, \ldots, H_{ip}\sigma, B_{i1}\sigma, \ldots, B_{iq}\sigma\}:
   \]
   \[
   \#D_i \supset \#\{B_{i1}\sigma, \ldots, B_{iq}\sigma\}.\]

2. For a propagation rule: $H_1, \ldots, H_n \Rightarrow G \mid B_1, \ldots, B_m$, with body-CHR constraints $B_k, \ldots, B_m$:
   \[
   \forall i \in \{1, \ldots, n\}, \forall j \in \{k, \ldots, m\} : |H_i|^\alpha \succ |B_j\sigma|^\alpha.\]

Observe that we still have substitutions in the formulation of this condition. But, we can avoid reasoning about them explicitly in an automation. It is enough to reason about all instances that satisfy the interargument relations.

**Theorem 2 (Sufficiency of the RC).** Let program $P$ with query $I$ satisfy the RC for CHR w.r.t. $|.|^\alpha$ and some associated interargument relations for all built-in constraints in the program, then all computations for $P$ with query $I$ terminate.

To prove this theorem, one observes that the level value of a rigid constraint is a natural number instead of a polynomial and adapts the proof of Theorem 1. We refer to [13] for the full proof.

**Example 8.** We prove termination for the Fibonacci program with this RC.

As in Example 4, the query is $fib(n, m)$ with $n$ a ground term, representing a natural number in successor-notation and $m$ a free variable. The call set is $\{fib(n, m), fib(n_2, n_3) \mid n_1, n_2, n_3 \text{ ground terms, representing natural numbers and } m \text{ a free variable}\}$. We use the abstract level mapping: $|fib(n, m)|^\alpha = |n|^\alpha$.

The call set is rigid w.r.t. the chosen abstract level mapping. For the first rule, $\{fib(N, M1), fib(N, M2), fib(N, M1)\}$ is a $\geq$-maximal subset w.r.t. $|.|^\alpha$. With two of these constraints in the head of the rule and only one in the body, this rule satisfies the RC for CHR. The second and third rule trivially satisfy the RC for CHR because they have no CHR constraints in the body. The last rule is a propagation rule. To satisfy the RC for CHR, every constraint matching the head of the rule must be larger than every constraint added by the body of the rule.

\[
|fib(s(s(N))), M|^\alpha = 2 + N \succ |fib(s(s(N)), M1)|^\alpha = 1 + N \\
|fib(s(s(N))), M|^\alpha = 2 + N \succ |fib(s(N), M2)|^\alpha = N.\]

□
Because no interargument relations are needed for the Fibonacci example, another small example is given.

**Example 9 (Greatest common divisor - gcd).** We prove termination for gcd with the RC for CHR. This program calculates the greatest common divisor of a set of positive integer numbers for a query: \( gcd(n) \ (n \in \mathbb{N}) \).

\[
gcd(0) \iff true.
gcd(M) \setminus gcd(N) \iff N >\geq M, M > 0, NN is N − M \mid gcd(NN).
\]

The call set is \{gcd(n) \mid n a natural number\}. As an abstract norm, we map each natural number or variable to itself. The associated interargument relations for the guard of the second rule are obvious (e.g. \{(n, m) \in \mathbb{N}^2 \mid n \geq m\} for \( N >\geq M \)). For any instance \( gcd(m) \setminus gcd(n) \iff n >\geq m, m > 0, nn is n − m \mid gcd(nn) \) satisfying the interargument relations of the guard, we have \( n > nn \).

The abstract level mapping \( |gcd(n)|^a = n \), proves termination w.r.t. the associated interargument relations and the chosen norm.

The call set is rigid w.r.t. the abstract level mapping. Because the first rule has no CHR constraints in the body, it trivially satisfies the RC with \{gcd(0)\} as a \( \geq \)-maximal subset. The second rule replaces a constraint \( gcd(n) \), mapped to \( n \), by the constraint \( gcd(nn) \), mapped to \( mn \). Since the associated interargument relations for this rule need to be satisfied: \( n > nn \). Therefore, \{gcd(N)\} is the only \( \geq \)-maximal subset of the rule. Since both rules satisfy the RC, this program terminates for the considered queries. 

\[\square\]

### 4 Automation and experimental evaluation

We have implemented a prototype analyzer to allow us to experimentally evaluate our approach. The prototype is implemented in CHR, with SWI-Prolog as host-language. Space restrictions do not allow us to give a full account of the system. In this section we focus on the most central components only and illustrate the general strategy with an example.

**Example 10 (Employee availability).** This program computes a list of all employees who have no appointment on a certain time point.

\[
app(Tb, Te, E), empl(L), time(T) \Rightarrow member(E, L), Tb \leq T, T \leq Te \mid remove(E).
empl(L_1), remove(E) \iff select(E, L_1, L_2) \mid empl(L_2).
\]

Here, \( select(e, l_1, l_2) \) only succeeds if the list \( l_1 \) contains \( e \) and \( l_2 \) is the list obtained from \( l_1 \) by removing one occurrence of \( e \).

The intention is that a query consists of constraints \( app/3 \), with a begin time, end time and employee’s name, representing appointments, a constraint \( empl/1 \), with as only argument the list of all employee’s names and a constraint \( time/1 \), with as its argument the time point on which we want the list of available employees. The propagation rule determines which employees need to be removed from the list. The simplification rule deletes employees from the list. The program terminates for all considered queries. 

\[\square\]
To prove termination of a CHR program with the RC for CHR, one has to find a norm, associated interargument relations and an appropriate abstract level mapping, such that the RC is satisfied. We use abstract linear level mappings. In the philosophy of the constraint based approach to termination analysis, as described in [3], we introduce a symbolic form of the level mapping.

Example 11 (Employee availability continued). For the constraints in employee availability we have the symbolic forms:
\[
\begin{align*}
|\text{time}(t)|^\alpha &= \text{time}_0 + \text{time}_1 \cdot |t|^\alpha, \\
|\text{app}(t_b, t_e, e)|^\alpha &= \text{app}_0 + \text{app}_1 \cdot |t_b|^\alpha + \text{app}_2 \cdot |t_e|^\alpha + \text{app}_3 \cdot |e|^\alpha, \\
|\text{empl}(e)|^\alpha &= \text{empl}_0 + \text{empl}_1 \cdot |e|^\alpha, \\
|\text{remove}(e)|^\alpha &= \text{remove}_0 + \text{remove}_1 \cdot |e|^\alpha.
\end{align*}
\]

Following the constraint based approach, the aim is to translate the conditions imposed by the RC into constraints of the symbolic coefficients (e.g. \text{app}_0, \text{app}_1, \text{app}_2, \text{app}_3) of the level mapping. This translation must be such that every solution for the resulting constraints corresponds to one way of satisfying the RC, and therefore, of obtaining a termination proof.

First, we perform a simple type inference to compute an overestimation of the call set. It is based on four basic types: \textit{ground list}, \textit{nil-terminated list}, \textit{ground term} and \textit{any term}. We initialize the call set with the given query type. Until a fix point is reached, constraints from the call set are matched with each rule’s head. For every applicable rule, the effect of the guard is analyzed and the call types of the CHR body constraints are added to the call set.

Example 12 (Employee availability continued). We initialize the call set as the set of all typed constraints in the query:
\[
\text{Call} = \{\text{app}(t_1, t_2, e), \text{empl}(l), \text{time}(t_3) | t_1, t_2, t_3, e \text{ ground terms and } l \text{ a ground list}\}.
\]

Only the first rule adds a new typed constraint, namely \(\text{remove}(e)\) with \(e\) a ground term. The fix point is reached and the call set is \(\text{Call} \cup \{\text{remove}(e) | \ e \text{ a ground term}\}\).

Next, we choose between term-size and list-length as the norm. The call set can provide us with information on how to select between them. But, the choice is backtrackable.

Example 13 (Employee availability continued). Due to the groundness of all terms in the call set, we first select term-size. It turns out that the analyzer fails for this choice. We then set the norm to list-length.

Now, we impose rigidity of the level mapping on the call set. We also ensure that the level mapping only measures arguments of the appropriate type.

Example 14 (Employee availability continued). The constraints \text{app}/3, \text{time}/1 and \text{remove}/1 have no list-arguments. Thus, we measure no arguments:
\[
\begin{align*}
|\text{app}(t_b, t_e, e)|^\alpha &= \text{app}_0, \\
|\text{time}(t)|^\alpha &= \text{time}_0, \\
|\text{remove}(e)|^\alpha &= \text{remove}_0
\end{align*}
\]

The only argument of \text{empl}/1 is a ground list, so \(|\text{empl}(l)|^\alpha = \text{empl}_0 + \text{empl}_1 \cdot |l|^\alpha\) is rigid on the call set.
In a next step, interargument relations are inferred for the guards. Our prototype implementation contains predefined interargument relations under list-length and term-size for built-in predicates. It requires the user to provide interargument relations for other guards. This can easily be further automated in the future by using interargument relation inference from another analyzer (e.g. Polytool [9]).

Example 15 (Employee availability continued). For member/2 and =< /2 there are no ‘real’ interargument relations under list-length, so that the relations are \(\mathbb{N}^2\). For select/3, the associated interargument relation is \(\{(n_1,n_2,n_3) \in \mathbb{N}^3 \mid n_2 = n_3 + 1\}\). □

We then reach the most central part of the system. Here, we set up constraints on the remaining symbolic coefficients of the level mapping, corresponding to the conditions expressed in the RC.

For propagation rules, this is fairly easy: provided that interargument relations hold of the guards, the level mapping should decrease from head to body constraints.

Example 16 (Employee availability continued). For the propagation rule, since the interargument relations for the guard are trivial, we get:

\[
\begin{align*}
&\text{app}(T_b,T_c,E)^\alpha > |\text{remove}(E)|^\alpha \quad \text{app}_0 > \text{remove}_0 \\
&\text{time}(T)^\alpha > |\text{remove}(E)|^\alpha \quad \text{or} \quad \text{time}_0 > \text{remove}_0 \\
&|\text{empl}(L_1)|^\alpha > |\text{remove}(E)|^\alpha \quad \text{empl}_0 + \text{empl}_1 > \text{remove}_0 \quad (1)
\end{align*}
\]

For simplification and simpagation rules this step is more difficult. The problem is that we need to reason about \(\ge\)-maximal subsets of constraints but that these subsets depend on the used level mapping – which, as yet, has not been fixed.

To solve this, we will compute all the possibilities of candidate \(\ge\)-maximal subsets, which are such that the condition in the RC is fulfilled. For each of these possibilities, we then express what conditions on the level mapping are required to make these subsets the \(\ge\)-maximal ones. If any of these conditions on the level mapping can be satisfied, the RC holds for the rule.

Example 17 (Employee availability continued). In the first column of Table 1, we present all the different cases of possible \(\ge\)-maximal subsets for the simplification rule, which are such that the RC is fulfilled. The second column of the table contains the corresponding condition on the level mapping needed for \(\ge\)-maximality of these sets.

<table>
<thead>
<tr>
<th>possible (\ge)-maximal subsets</th>
<th>Resulting set of equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>{empl(L)} {remove(E)} {empl(L_z)} {remove(E)}</td>
<td></td>
</tr>
<tr>
<td>{empl(L_1)} {remove(E)}</td>
<td></td>
</tr>
<tr>
<td>{empl(L_1)} {remove(E)} {empl(L_z)} {remove(E)}</td>
<td></td>
</tr>
<tr>
<td>{empl(L_1)} {remove(E)} {empl(L_z)} {remove(E)}</td>
<td></td>
</tr>
<tr>
<td>{empl(L_1)} {remove(E)} {empl(L_z)}</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. \(\ge\)-maximal subsets for employee availability
Observe that for the four last rows, the resulting equations are inconsistent with the constraint (1) obtained for the propagation rule. For instance, for the second row, $|\text{remove}(E)|^\alpha > |\text{empl}(L_1)|^\alpha$ is inconsistent with $|\text{empl}(L_1)|^\alpha > |\text{remove}(E)|^\alpha$.

As a result, the first row is the only remaining candidate constraint. Since the interargument relation of the guard in the simplification rule is non-trivial, we obtain the condition:

\[
L_1 = L_2 + 1 \implies |\text{empl}(L_1)|^\alpha > |\text{empl}(L_2)|^\alpha \wedge |\text{empl}(L_1)|^\alpha > |\text{remove}(E)|^\alpha
\]

or $L_1 = L_2 + 1 \implies \text{empl}_0 + \text{empl}_1.L_1 > \text{empl}_0.L_2 + \text{empl}_1.L_2 \wedge \text{empl}_0 + \text{empl}_1.L_1 > \text{remove}_0$

Finally, all conditions on the symbolic coefficients are collected and transformed into Diophantine constraints, by using the method described in [9]. These equations are solved using the constraint solver of Aprove ([6]).

Example 18 (Employee availability continued).
Aprove finds the solution $\text{app}_0 = \text{empl}_0 = \text{time}_0 = \text{empl}_1 = 1$, $\text{remove}_0 = 0$, corresponding to the level mapping:

\[
\begin{align*}
|\text{app}(Tb, Te, E)|^\alpha & = 1 \\
|\text{empl}(L)|^\alpha & = 1 + ||L||^\alpha \\
|\text{remove}(E)|^\alpha & = 0 \\
|\text{time}(T)|^\alpha & = 1
\end{align*}
\]

4.1 Experimental evaluation

There are some benchmarks available in the CHR community. Because they are aimed at testing performance issues, they are relatively small and not useful for testing a termination analyzer. Therefore, we collected a number of programs from various sources to set up a new benchmark. It contains 52 programs: 39 without propagation and 13 with propagation. All programs are terminating.

We compared our prototype analyzer with the transformational analyzer of [10], which can not deal with propagation. The results are very satisfactory. Our analyzer proves termination for 31 programs without propagation. The analyzer of [10] proves termination of 26 of these programs. The transformational analyzer proves termination of two programs where our approach fails. This is because these programs need a more complex norm as list-length or term-size. The results are presented in Tables 2 and 3 in an appendix for the referees.

Our implementation proves 10 programs with propagation from the benchmark terminating. The failing programs increase an arguments size till a certain bound. Our approach cannot deal with bounded increases. For term rewrite systems, a technique to prove termination for such programs is discussed in [7].

5 Conclusion

In this paper we discussed a new approach to termination analysis of CHR programs. To date, automated termination analysis was restricted to CHR programs with simplification only. Our condition allows for termination analysis of general CHR programs, that is, CHR programs with propagation as well. We have
implemented the technique in an automated system. Experimental results with this system show that it is successful in proving termination for a majority of the test set programs.

The condition on simplification rules, as proposed in [5], was strengthened in our RC in order to be able to extend it with a condition for propagation rules. Therefore, a small class of CHR programs cannot be proved terminating with our approach, where the (non-automated) approach of [5] succeeds.

Future work will be directed towards scalability of our approach. We intend to adapt the dependency pair method, which has proved very successful both for TRS [6] and for LP [8], to the CHR context.

References