

A new approach to termination analysis of CHR

Dean Voets Paolo Pilozzi Danny De Schreye

Report CW 506, January 2008



Katholieke Universiteit Leuven
Department of Computer Science

Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

A new approach to termination analysis of CHR

Dean Voets Paolo Pilozzi Danny De Schreye

Report CW 506, January 2008

Department of Computer Science, K.U.Leuven

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 [7], 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, which replace constraints by simpler constraints, *propagation* rules, which add logically redundant constraints, and *simpagation* rules, which are a combination of both. Its simple syntax and semantics make it well-suited for implementing custom constraint solvers [7, 13, 6, 14, 15]. Particularly the latter feature of the language, accounts for its success and impact on the research community.

Although the language is strongly related to Logic Programming (LP) and to a lesser extent also to Term-Rewrite Systems (TRS), termination analysis of CHR programs has received little attention. To the best of our knowledge, there are only two contributions to date. The main contribution is reported in [8] and is limited to CHR programs without propagation rules. The author shows that, for this class of CHR programs, termination analysis techniques developed for LP [3] and TRS [5] are adaptable to the CHR context.

The second contribution is a transformational approach, in which CHR programs are transformed to equivalent Prolog programs [12] and analyzed with termination analysis tools for Prolog. This approach is restricted to programs without multi-headed propagation rules.

Our approach is different in that we verify whether only a finite number of constraints can enter the constraint store during any computation. As such, we can consider the propagation history, which allows us to prove termination for programs with propagation rules. We implemented the method in a prototype analyzer and performed an experimental evaluation. The results were very satisfactory. Since the approach presented in [8] was not implemented, this analyzer is the first non-transformational automated termination analyzer for CHR.

The paper is organized as follows. In the next section, we introduce the essential aspects of CHR and adapt some basic concepts from termination analysis of LP to the CHR context. In Section 3, we introduce a ranking condition for general CHR programs that is sufficient for proving termination. We further refine the condition so that it can be automated. In Section 4 and Section 5, we discuss some possible optimizations to this approach and our prototype analyzer, respectively. Finally Section 6 concludes this 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 solver, and *CHR constraints*, user-defined and solved by a *CHR program*. A CHR program is a finite set of *CHR rules*. Rules are of the form:

Simplification rule:	Propagation rule:	Simpagation rule:
$[N@] true \setminus H \Leftrightarrow [G \mid] B.$	$[N@] H \setminus true \Leftrightarrow [G \mid] B.$	$[N@] H_1 \setminus H_2 \Leftrightarrow [G \mid] B.$
or $[N@] H \Leftrightarrow [G \mid] B.$	or $[N@] H \Rightarrow [G \mid] B.$	

A rule has an optional name N . The head H (H_1 and 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.

Example 1 (Fibonacci). Mathematically, the Fibonacci function is defined as follows, where $n \in \mathbb{N}$.

$$fib(n) = \begin{cases} \text{if } n \leq 1 \text{ then } 1 \\ \text{if } n > 1 \text{ then } fib(n-1) + fib(n-2) \end{cases}$$

The CHR program below implements a Fibonacci algorithm. Natural numbers are written in the symbolic notation, in which zero is written as 0 and all other numbers are depicted with the successor function s . In this program, `add/3` is a built-in constraint which defines addition on natural numbers written in the symbolic notation.

$$\begin{aligned} fib(N, M1), fib(N, M2) &\Leftrightarrow M1 = M2, fib(N, M1). \\ fib(0, M) &\Rightarrow M = s(0). \\ fib(s(0), M) &\Rightarrow M = s(0). \\ fib(s(s(N)), M) &\Rightarrow fib(s(N), M1), fib(N, M2), add(M1, M2, M). \end{aligned}$$

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. \square

Operational Semantics. A CHR program defines a state transition system, where the *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. In a *final state* or *answer*, either the built-in constraints are inconsistent (*failed state*), or no more transitions are possible.

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

$$\begin{aligned} H'_1 \wedge H'_2 \wedge D &\mapsto ((H_1 \wedge H_2) = (H'_1 \wedge H'_2)) \wedge G \wedge B \wedge H'_1 \wedge D \\ \text{if } H_1 \setminus H_2 &\Leftrightarrow G \mid B. \text{ in } P \text{ and} \\ CT \models D &\rightarrow \exists \theta, \theta' ((H_1 \theta \wedge H_2 \theta) = (H'_1 \wedge H'_2)) \wedge G \theta \theta' \end{aligned}$$

A rule is applicable to a conjunction of CHR constraints, H' , if these match the head atoms H_1 and H_2 of the rule, with matching substitution θ , such that the guard G evaluates to true, with answer substitution θ' , given the built-ins in the store. B is a conjunction of built-in and CHR constraints, added to the constraint store. Rule application is non-deterministic and committed-choice. \square

These semantics are called the *theoretical semantics*. For more information about the operational semantics of CHR, we refer to [7].

A CHR program P with query I *terminates*, if all computations for P with query I end in a final or failed state. Because propagation rules do not remove constraints from the store, a fire once policy is used to prevent trivial non-termination. The *propagation history* implements this policy. For a more detailed description of the propagation history, we again refer to [7].

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 $fib(s(s(s(0))), N)$, the last propagation rule adds two new $fib/2$ 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 $fib(s(s(0)), N)$, 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

$$fib(s(s(s(0))), s(s(s(0)))) \wedge fib(s(s(0)), s(s(0))) \wedge fib(s(0), s(0)) \wedge fib(0, s(0))$$

is the final state. Without the simplification rule the answer would contain an additional $fib(s(0), s(0))$ constraint.

Note that the execution of this program becomes non-deterministic for larger Fibonacci numbers, because higher duplicates can fire the last rule before being removed by the simplification rule. \square

2.2 Termination Analysis

Termination analysis for logic programs is usually done by showing a decrease between consecutive computation states. Atoms are mapped to natural numbers using a *norm* and *level mapping*[4]. A norm is a function which maps terms to natural numbers. A level mapping is a function which maps atoms to natural numbers. The sizes of consecutive computation states are compared using the *level values* of the head and body atoms of the rules of a CHR program. We redefine these functions in the CHR context.

Definition 2 (norm, level mapping). A *norm* is a mapping $||\cdot|| : Term_P \rightarrow \mathbb{N}$. A *level mapping* is a mapping $|\cdot| : Con_P \rightarrow \mathbb{N}$. \square

We will refer to $|C|$ as the level value of C . Here, Con_P are all CHR constraints defined by a CHR program P with arguments of $Term_P$, the set of all terms constructible from P .

Several examples of norms and level mappings can be found in the literature on LP termination analysis [3]. Two well-known norms are *list-length* and *term-size*. The most common kind of level mapping is the *linear level mapping*.

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

List-length is defined as:

$$\begin{aligned} \|[t_1 t_2]\|_l &= 1 + \|t_2\|_l \text{ with } t_1 \text{ and } t_2 \text{ any term,} \\ \|t\|_l &= 0 \quad \text{otherwise.} \end{aligned}$$

Term-size is defined as:

$$\begin{aligned} \|f(t_1, t_2, \dots, t_n)\|_t &= 1 + \sum_{1 \leq i \leq n} \|t_i\|_t \text{ with } f \text{ any function symbol and } n > 0 \\ \|t\|_t &= 0 \quad \text{otherwise.} \end{aligned}$$

□

Term-size can be used to compare natural numbers written in their symbolic notation.

Definition 4 (linear level mapping).

A *linear level mapping* is any level mapping which can be defined as:

$$\begin{aligned} |con(t_1, \dots, t_n)| &= con_0 + \sum_{1 \leq i \leq n} con_i \times \|t_i\|, \\ \text{with } con_i \in \mathbb{N} &\text{ only depending on } con \text{ and } \|\cdot\| \text{ a norm.} \end{aligned}$$

□

Example 3 (Linear level mappings).

$$\begin{aligned} |mergesort([A, B, C])| &= 0 + 1 \times \|[A, B, C]\|_l = 3 \\ |fib(s(s(0)), M)| &= 0 + 1 \times \|s(s(0))\|_t + 0 \times \|M\|_t = 2 \end{aligned}$$

□

As stated, proving termination is usually done by proving a decrease, w.r.t. a level mapping, for every rule of a program. If such a decrease is proven, the program terminates for every *bounded* query w.r.t. that level mapping.

Definition 5 (bounded constraint).

A CHR constraint C is **bounded** w.r.t. level mapping $|\cdot|$, iff:

$$\exists max \in \mathbb{N} : \forall \sigma : |C\sigma| \leq max$$

with σ a substitution. A conjunction of constraints is bounded w.r.t. a level mapping, if every CHR constraint in the conjunction is bounded w.r.t. that level mapping. □

Ground CHR constraints and ground terms do not contain variables and are bounded w.r.t. any level mapping and norm, respectively.

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 , $\mathbf{Call}(P, I)$, is the set of all CHR-constraints which are added to the constraint store during a computation of P for I . □

Usually, we will specify $\mathbf{Call}(P, I)$ using an abstraction. The abstraction will be such that every CHR constraint which is added to the store, during an execution of program P with query I , is an instance of an element in $\mathbf{Call}(P, I)$.

2.3 Ranking condition for CHR with simplification only

In [8], 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 all CHR rules in the program. Linear ranking functions, functions very similar to linear level mappings, are used to map conjunctions of constraints to natural numbers.

A ranking condition is formulated, which imposes certain conditions on the ranking function w.r.t. the program. If the ranking condition holds for a program, then this program terminates for every bounded goal.

Definition 7 (Ranking condition (RC) for simplification only [8]). *The ranking condition for simplification only is the formula $\forall(RC(G, B) \rightarrow H \succ B)$ for all rules $H' \setminus H \Leftrightarrow G \mid B$, where $RC(G, B)$ is the conjunction of the rank constraints derived from the built-ins in the guard and body of the rule. \square*

By $H \succ B$ we mean $rank(H) > rank(B)$, with $rank$ a ranking function. Variables are mapped to natural numbers greater or equal than 0. The theorem given below is proved.

Theorem 1 (Sufficiency of the RC for simplification only [8]). *Given a CHR program P with only rules of the simplification type, and a ranking where:*

- $rank((A \wedge B)) = rank(A) + rank(B)$ for any two constraints A and B ,
- or $rank((A \wedge B)) = rank(A) \uplus rank(B)$ for any two constraints A and B , with \uplus the multiset union.

If the ranking condition holds for each rule in P , P is terminating for all bounded queries. \square

The next simple example illustrates the RC for CHR with simplification only.

Example 4 (Addition). The next program adds a set of integers and counts the number of additions that were performed.

$$\begin{aligned} count(C), int(N1), int(N2) \Leftrightarrow N \text{ is } N1 + N2, CC \text{ is } C + 1, \\ count(CC), int(N). \end{aligned}$$

The program obviously terminates as the number of constraints in the constraint store, used to represent the integers, is decreased every time the rule fires. We use a linear ranking function to map constraints to natural numbers. With coefficients,

$$\begin{array}{lll} a_0^{count/1} = 1 & a_1^{count/1} = 0 & |count(C)| = 1 + 0 \times ||C|| = 1 \\ a_0^{int/1} = 1 & a_1^{int/1} = 0 & |int(N)| = 1 + 0 \times ||N|| = 1 \end{array}$$

Thus, measuring the size of the constraint store by the sum of measures of the CHR constraints in it, we get $3 > 2$ or, when using multi-set order, $\{1, 1, 1\} >_m \{1, 1\}$. Both prove termination of the CHR program for all queries as all queries are bounded w.r.t. $|\cdot|$. \square

3 A new ranking condition for CHR

The extension to programs with propagation rules gives a totally new termination problem. In LP, TRS and CHR with simplification only, a decrease between consecutive states implies termination. Here, for LP and TRS we refer to decreases of sizes of atoms or terms, while for *CHR with simplification only* we refer to decreases of the sizes of constraints in the store. For CHR with propagation, new constraints are added and no existing constraints are removed. 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.

The next Lemma states that if only a finite number of CHR constraints are added to the constraint store, program P with query I terminates. This Lemma is used to prove sufficiency of our ranking conditions. 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 , all executions of P for I only execute a finite number of rules. Therefore, only a finite number of CHR constraints are added to the store during any execution of P for I .

\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 executed a finite number of times. Since every rule can only be executed a finite number of times, P terminates for I . \square

To prove termination of general CHR programs, a ranking condition is proposed which implies termination by a finite addition of CHR constraints. The condition given below is not suited for automation. We will later refine this condition so that it can be automated.

3.1 Ranking Condition (RC) for CHR with substitutions

The next definition gives the first version of our ranking condition. It is applicable to general CHR programs and defines when rules in programs satisfy the RC.

Definition 8 (Ranking condition for CHR with substitutions). A program P and a query I satisfy the RC for CHR, w.r.t. level mapping $|\cdot|$ iff every CHR constraint in $\text{Call}(P, I)$ is bounded w.r.t. $|\cdot|$ and for each rule in P , and for every matching substitution θ and answer substitution θ' from Definition 1:

1. For a simplification or simpagation rule $H \setminus H_1, \dots, H_n \Leftrightarrow G \mid B_1, \dots, B_m$, with body-CHR constraints B_k, \dots, B_m , then

let $p = \max \{|H_1\theta|, \dots, |H_n\theta|, |B_k\theta\theta'|, \dots, |B_m\theta\theta'|\}$
 The number of CHR constraints with level value p has to be higher in $\{H_1\theta, \dots, H_n\theta\}$ than in $\{B_k\theta\theta', \dots, B_m\theta\theta'\}$
2. For a propagation rule: $H_1, \dots, H_n \Rightarrow G \mid B_1, \dots, B_m$, with body-CHR constraints B_k, \dots, B_m , then

for all $i = 1, \dots, n$ and $j = k, \dots, m$: $|H_i\theta| > |B_j\theta\theta'|$. □

This ranking condition implies that only a finite number of CHR constraints are added to the constraint store during the execution of P for I . Therefore, Lemma 1 proves termination for all programs and queries which satisfy this ranking condition.

Theorem 2 (Sufficiency of the RC with substitutions). Let program P with query I satisfy the RC with substitutions w.r.t. $|\cdot|$, then all computations for P with query I terminate. □

Proof. In order to prove termination of a CHR program P with a query I , it is sufficient to prove that the total number of CHR constraints, added during an execution of P for I , is finite. We will prove this using induction.

Base case. The query is bounded w.r.t. the level mapping $|\cdot|$, so there exists a maximal level value of the CHR constraints in the query, max . Because of the ranking condition, only simplification or simpagation rules can add constraints with level value max to the store. 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 an execution of P for I .

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

- The query 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}, \dots, 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 , $i : n \leq i \leq max$, 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 if program P and query I satisfy the RC then only a finite number of CHR constraints are added to the store during an execution of P for I . Therefore, Lemma 1 proves termination. \square

Example 5 (Fibonacci continued). We prove termination for the Fibonacci example with this ranking condition.

Let $fib(N, M)$ 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 $\{fib(N_1, M), fib(N_2, N_3) \mid N_1, N_2, N_3 \text{ ground terms, representing natural numbers and } M \text{ a free variable}\}$. As a norm, we use term-size. The level mapping is defined on the call set as $|fib(N, M)| = ||N||_t$. Clearly, the call set is bounded w.r.t. $|\cdot|$.

For the first rule we have that for every matching substitution θ , the first term in every fib/2 constraint is substituted by the same ground term. The answer substitutions θ' are empty because this rule has no guard.

$$|fib(N, M1)\theta| = |fib(N, M2)\theta| = |fib(N, M1)\theta| = ||N||_t, \text{ with } N \text{ a ground term.}$$

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. Because the second and third rule have no CHR constraints in the body, these rules trivially satisfy the RC for CHR with substitutions.

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

$$\begin{aligned} |fib(s(s(N)), M)\theta| &= 2 + ||N||_t > |fib(s(N), M1)\theta\theta'| = 1 + ||N||_t, \text{ with } N \text{ ground,} \\ |fib(s(s(N)), M)\theta| &= 2 + ||N||_t > |fib(N, M2)\theta\theta'| = ||N||_t, \text{ with } N \text{ ground.} \end{aligned}$$

For all matching substitutions and answer substitutions, the ranking condition is satisfied. Therefore, the program is termination w.r.t. the query. \square

3.2 Ranking Condition for CHR

A disadvantage of the ranking condition of Definition 8, is that one has to regard all matching and answer substitutions. This cannot be done automatically because in general there are infinitely many matching and answer substitutions. In this section, we present a ranking condition which is suitable for automation. In order to estimate the effects of the matching substitutions, *abstract norms* and *abstract level mappings* are adapted from [4] and [10]. 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.

Definition 9 (abstract norm, abstract level mapping). An *abstract norm* is a mapping $\|\cdot\| : \text{Term}_P \rightarrow \mathbb{N}[\text{Var}_P]$, which is the identity function on Var_P . An *abstract level mapping* is a mapping $|\cdot| : \text{Con}_P \rightarrow \mathbb{N}[\text{Var}_P]$. \square

Here, $\mathbb{N}[\text{Var}_P]$ denotes the set of polynomials over Var_P , with natural coefficients.

Definition 10 (abstract list-length, abstract term-size). *Abstract list-length* is defined as:

$$\begin{aligned} \|[t_1|t_2]\|_l^\alpha &= 1 + \|t_2\|_l^\alpha \text{ with } t_1 \text{ and } t_2 \text{ any term,} \\ \|t\|_l^\alpha &= t \text{ with } t \text{ a variable,} \\ \|t\|_l^\alpha &= 0 \text{ otherwise.} \end{aligned}$$

Abstract term-size is defined as:

$$\begin{aligned} \|f(t_1, t_2, \dots, t_n)\|_t^\alpha &= 1 + \sum_{1 \leq i \leq n} \|t_i\|_t^\alpha \text{ with } t_1 \text{ and } t_2 \text{ any term,} \\ \|t\|_t^\alpha &= t \text{ with } t \text{ a variable,} \\ \|t\|_t^\alpha &= 0 \text{ otherwise.} \end{aligned}$$

\square

The only difference with list-length and term-size of Definition 3 is that variables are mapped to themselves instead of to zero.

Definition 11 (abstract linear level mapping).

An *abstract linear level mapping* is any mapping which can be defined as:

$$\begin{aligned} |\text{con}(t_1, \dots, t_n)|^\alpha &= \text{con}_0 + \sum_{1 \leq i \leq n} \text{con}_i \times \|t_i\|^\alpha, \\ &\text{with } \text{con}_i \in \{0, \dots, n\} \text{ only depending on } \text{con} \text{ and } \|\cdot\|^\alpha \text{ an abstract norm.} \end{aligned} \quad \square$$

Example 6 (Abstract linear level mappings).

$$\begin{aligned} |\text{mergesort}([L|\text{List}])|^\alpha &= 0 + 1 \times \|[L|\text{List}]\|_l^\alpha = 1 + \text{List} \\ |\text{fib}(s(s(N)), M)|^\alpha &= 0 + 1 \times \|s(s(N))\|_t^\alpha + 0 \times \|M\|_t^\alpha = 2 + N \end{aligned} \quad \square$$

In general, an abstract level mapping maps constraints to arbitrary polynomials over \mathbb{N} . We use abstract linear level mappings with abstract list-length or abstract term-size as abstract norms. As a result, we get linear polynomials. In order to compare the level values of the constraints w.r.t. an abstract level mapping, we define an ordering on polynomials over \mathbb{N} [10].

Definition 12 (orderings on $\mathbb{N}[\text{Var}_P]$). Let p and q be two polynomials. Let X_1, \dots, X_n be all variables occurring in p or q . The quasi-ordering \succeq is defined as $p \succeq q$ iff $p(x_1, \dots, x_n) \geq q(x_1, \dots, x_n)$ for all $x_1, \dots, x_n \in \mathbb{N}$. The strict ordering \succ is defined as $p \succ q$ iff if $p(x_1, \dots, x_n) > q(x_1, \dots, x_n)$ for all $x_1, \dots, x_n \in \mathbb{N}$. The equality between polynomials is defined as $p \approx q$ iff $p(x_1, \dots, x_n) = q(x_1, \dots, x_n)$ for all $x_1, \dots, x_n \in \mathbb{N}$. \square

The next example shows the orderings between three polynomials.

Example 7 (polynomial ordering).

Let $p(X, Y) = 1 + XY + 2X$, $q(X) = 2X$ and $z(X, Y) = XY + X$:

- $p(X, Y) \succ q(X)$, $p(X, Y) \succeq q(X)$.
- $p(X, Y) \succ z(X, Y)$, $p(X, Y) \succeq z(X, Y)$
- 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 13 (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 $R_p \in \mathbb{N}^n$. R_p is a valid interargument relation for p/n w.r.t. a norm $\|\cdot\|$, iff*

$$\forall t_1, \dots, t_n \in Term_P : CT \models p(t_1, \dots, t_n) \implies (\|t_1\|, \dots, \|t_n\|) \in R_p. \quad \square$$

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

Example 8.

built-in constraint	norm and interargument relation
delete(N1,N2,N3)	list-length: $\{(n_1, n_2, n_3) \in \mathbb{N}^3 \mid n_1 \geq n_3\}$
append(N1,N2,N3)	list-length: $\{(n_1, n_2, n_3) \in \mathbb{N}^3 \mid n_1 + n_2 = n_3\}$
leq(N1,N2)	term-size: $\{(n_1, n_2) \in \mathbb{N}^2 \mid n_1 \leq n_2\}$

The built-in constraint delete/3 removes the elements unifying with N2 from list N1. N3 is the resulting list. The built-in constraint append/3 appends lists N1 and N2. The result is N3. For both these examples we use the list-length norm. The last built-in constraint is a built-in representing the \leq relation on numbers in the successor-notation. Here, we use term-size as a norm. □

Because we do not consider each matching and answer substitution, we need a stricter notion as boundedness. A substitution should not change the level value of a constraint in the constraint store w.r.t. the abstract level mapping. If a CHR constraint satisfies this condition, we call this constraint *rigid* w.r.t. this abstract level mapping.

Definition 14 (Rigidity). *A CHR constraint C is **rigid** w.r.t. an abstract level mapping $|\cdot|^\alpha$ iff \forall substitutions $\theta : |C|^\alpha \approx |C\theta|^\alpha$. □*

An alternative definition is given in [10], the author characterizes rigidity with *relevant variables*. This definition shows that the value of a rigid constraint w.r.t. an abstract level mapping is a natural number instead of a polynomial.

Definition 15 (relevant variables). *Let $|\cdot|^\alpha$ be an abstract level mapping and A be a CHR constraint. A variable X in A is called relevant w.r.t. $|\cdot|^\alpha$ if there exists a substitution σ for X such that $|A\sigma|^\alpha \not\approx |A|^\alpha$.*

Example 9 (relevant variables). Let $A = p([X|Y])$ and $|p(X)|^\alpha = \|X\|_l^\alpha$, i.e., $\|[X|Y]\|_l = 1 + \|Y\|_l$. Then the only relevant variable of A is Y . □

Proposition 1 (alternative characterization of rigidity). *Let $|\cdot|^\alpha$ be an abstract level mapping and A be a CHR constraint. Then A is rigid w.r.t. $|\cdot|^\alpha$ iff A has no relevant variables w.r.t. $|\cdot|^\alpha$.*

Proof. Obvious from Definitions 14 and 15. \square

In our ranking condition with substitutions as in Definition 8, we checked a condition on *instances of rules*: rules with a matching and answer substitution. As stated, the effect of the matching substitution are estimated using an abstract level mapping, that maps CHR constraints to polynomials. For simplification and simpagation rules in the ranking condition of Definition 8, we used the maximum p of the level values of the constraints in the rule. When using an abstract level mapping, we need a similar notion for polynomials.

Definition 16 (\succeq -maximal subset of constraints).

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

- *The constraints in D have the same level value:*
 $\forall \text{ CHR-constraints } C1, C2 \in D : |C1|^\alpha \approx |C2|^\alpha$
- *There are no bigger constraints in $C \setminus D$.*
 $\nexists C1 \in D, \exists C2 \in C \setminus D : |C2|^\alpha \succeq |C1|^\alpha$

We refine our ranking condition from Definition 8 using abstract level mappings and interargument relations.

Definition 17 (Ranking condition for CHR).

A program P and a query I satisfy the RC for CHR, w.r.t. an abstract level mapping $|\cdot|^\alpha$ iff every constraint in $\text{Call}(P, I)$ is rigid w.r.t. $|\cdot|^\alpha$ and for each rule in P , and for each substitution σ such that the built-in constraints in the guard all satisfy their associated interargument relations:

1. *For a simplification or simpagation rule $H \setminus H_1, \dots, H_n \Leftrightarrow G \mid B_1, \dots, B_m$, with body-CHR constraints B_k, \dots, B_m and D_1, \dots, D_c all \succeq -maximal subsets of $\{H_1, \dots, H_n, B_k\sigma, \dots, B_m\sigma\}$, then*

$$\forall_{i \in \{1, \dots, c\}} D_i = \{H_{i_1}, \dots, H_{i_p}, B_{i_1}\sigma, \dots, B_{i_q}\sigma\}:$$

$$\#\{H_{i_1}, \dots, H_{i_j}\} > \#\{B_{i_1}\sigma, \dots, B_{i_i}\sigma\}$$
2. *For a propagation rule: $H_1, \dots, H_n \Rightarrow G \mid B_1, \dots, B_m$, with body-CHR constraints B_k, \dots, B_m , then*

$$\forall i \in \{1, \dots, n\}, \forall j \in \{k, \dots, m\} : |H_i|^\alpha \succ |B_j\sigma|^\alpha. \quad \square$$

This ranking condition can be automated, since we only have to consider a finite number of rules and interargument relations. The RC for CHR implies termination due to a finite addition of CHR constraints.

Theorem 3 (Sufficiency of the RC). *Let program P with query I satisfy the RC for CHR w.r.t. $|\cdot|^\alpha$ and some associated interargument relations for all built-in constraints in the program, then all computations for P with query I terminate. \square*

Proof. In order to prove termination of a CHR program P with a query I , it is sufficient to prove that the total number of CHR constraints, added during an execution of P with query I , is finite. We will prove this using induction.

Since the call set is rigid w.r.t. the level mapping $|\cdot|^\alpha$, each CHR constraint C , which is added to the constraint store by the query or an instance of a rule, has no relevant variables w.r.t. $|\cdot|^\alpha$. Therefore, $|C|^\alpha$ is a natural number.

Base case. The query is rigid w.r.t. $|\cdot|^\alpha$, so there exists a maximal level value of the CHR constraints in the query, max . Because of the ranking condition, each instance of a propagation rule can only add constraints with a level value lower than max and each simplification or simpagation rule can only add CHR constraints with a level value max or lower to the store.

If program P with query I satisfies the RC for CHR with $|\cdot|^\alpha$, all CHR constraints which are added to the store during any computation of P with query I have a natural number as level value w.r.t. $|\cdot|^\alpha$ and there is a maximal level value max .

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

- The query only contains a finite number of constraints with level value n .
- For each propagation rule which satisfies the RC w.r.t. $|\cdot|^\alpha$, every head constraint has a larger level value, w.r.t. \succ , than every CHR constraint in the body.

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 natural number larger than n as a level mapping. Because of the upper limits a_{max}, \dots, a_{n+1} and the propagation history, every propagation rule can only add a finite number of CHR constraints with level value n .

- For each simplification or simpagation rule which satisfies the RC w.r.t. $|\cdot|^\alpha$, every maximal subset of CHR constraints has more constraints from the head than from body of the rule. For every instance of a simplification or simpagation rule with maximal subsets D_1, \dots, D_n , there are maximally ranked subsets D_{i_1}, \dots, D_{i_p} , which contain the CHR constraints with the highest level value, m , w.r.t. $|\cdot|^\alpha$. Since the program satisfies the RC for CHR, the number of CHR constraints with level value m decreases and all CHR constraints added by the body of the rule have a level mapping smaller than or equal to m . This implies that only a finite number of constraints with level mapping n can be added to the store by instances of simpagation and simplification rules, because after enough rule executions, the constraint store has 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. \square

Example 10. We prove termination for the Fibonacci program with this ranking condition.

$$\begin{aligned}
& fib(N, M1), fib(N, M2) \Leftrightarrow M1 = M2, fib(N, M1). \\
& fib(0, M) \Rightarrow M = s(0). \\
& fib(s(0), M) \Rightarrow M = s(0). \\
& fib(s(s(N)), M) \Rightarrow fib(s(N), M1), fib(N, M2), add(M1, M2, M).
\end{aligned}$$

As in Example 5, the query is $fib(N, M)$ 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 $\{fib(N_1, M), fib(N_2, N_3) \mid N_1, N_2, N_3 \text{ ground terms, representing natural numbers and } M \text{ a free variable}\}$. We will use level mapping: $|fib(N, M)|^\alpha = ||N||_t^\alpha$.

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 \succeq -maximal subset w.r.t. $|\cdot|^\alpha$. 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.

$$\begin{aligned}
|fib(s(s(N)), M)|^\alpha &= 2 + N \succ |fib(s(N), M1)|^\alpha = 1 + N \\
|fib(s(s(N)), M)|^\alpha &= 2 + N \succ |fib(N, M2)|^\alpha = N.
\end{aligned}$$

Thus, this rule satisfies the RC for CHR as well. Since every rule of this program satisfies the RC for CHR, the program terminates for the considered queries. \square

Example 11 (Greatest common divisor - gcd). We prove termination for gcd with this ranking condition. This program calculates the greatest common divisor of a set of positive integer numbers for a query: $gcd(N)$ ($N \in \mathbb{N}$).

$$\begin{aligned}
& gcd(0) \Leftrightarrow true. \\
& gcd(M) \setminus gcd(N) \Leftrightarrow N \geq M, M > 0, NN \text{ is } N - M \mid gcd(NN).
\end{aligned}$$

The call set is $\{gcd(N) \mid N \text{ a natural number}\}$. 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:

$$\begin{aligned}
N \geq M & \quad \{(n, m) \in \mathbb{N}^2 \mid n \geq m\} \\
M > 0 & \quad \{(m) \in \mathbb{N} \mid m > 0\} \\
NN \text{ is } N - M & \quad \{(nn, n, m) \in \mathbb{N}^3 \mid nn = n - m\}
\end{aligned}$$

When all interargument relations are satisfied for the guard of the second rule, it follows that $N > NN$.

The abstract level mapping given below, proves termination w.r.t. the associated interargument relations and the chosen norm.

$$a_0^{gcd/1} = 0 \quad a_1^{gcd/1} = 1 \quad |gcd(N)|^\alpha = N$$

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 ranking condition with $\{gcd(0)\}$ as a \succeq -maximal subset. The second rule replaces a constraint $gcd(N)$, mapped to N , by the constraint $gcd(NN)$, mapped to NN . Since the associated interargument relations for this rule need to be satisfied, $|gcd(N)|^\alpha = N > |gcd(NN)|^\alpha = NN$ is implied. Therefore, $\{gcd(N)\}$ is the only \succeq -maximal subset of the rule. Since both rules satisfy the ranking condition, this program terminates for the considered queries. \square

4 Further Optimizations

The conditions of Definitions 7 8 and 17 are examples of *global approaches* to prove termination of CHR programs. In these approaches, one level mapping proves termination of a program. Another kind of approach, is a *local approach*, where different level mappings are used for different loops [2][1]. We adapt a technique based on *dependency graphs*, as presented in [11], to split our program into different components, which can be analyzed separately. The main advantage of this approach over a global approach, is efficiency. Another benefit is that different techniques to prove termination can be combined using this technique. For each rule r , the dependency graph shows us the rules matching the constraints added by r .

Definition 18 (dependency graph). A *dependency graph* for a program P , is a directed graph (V,E) . The graph has a node for each rule in P . The graph has an edge from *node1* to *node2*, if the body of the rule corresponding to *node1* has a constraint which matches a constraint in the head of the rule corresponding to *node2*.

We split our program into *components* using this dependency graph.

Definition 19 (component). Two rules r and s , of a CHR program P , are in the same *component* iff the dependency graph contains a path from r to s and a path from s to r .

The example given below explains the use of components to prove termination.

Example 12 (Mean).

This program, from [12], calculates the mean of a set of natural numbers in the symbolic notation. The constraint *mean* is used to start the calculation. The constraints $e/1$ represent the numbers of which the mean is calculated. Constraint $c/1$ counts the numbers in the set. Constraint $eC/1$ makes sure every number is used before the last rule calculates the mean. The built-in constraints $add/3$ and $div/4$ represent the addition and division of natural numbers in the symbolic notation.

$$\begin{aligned} r1 @ mean &\Leftrightarrow c(s(0)), mean(0). \\ r2 @ e(A), e(B), c(D) &\Leftrightarrow add(A, B, C), e(C), eC(C), c(s(D)). \\ r3 @ e(A), c(B), eC(A), mean(-) &\Leftrightarrow div(A, B, C, -), mean(C). \end{aligned}$$

The program is started with a number of $e/1$ constraints and a *mean* constraint. Rule $r1$ is executed first. Rule $r2$ is repeated until the sum of all numbers is found, then rule $r3$ calculates the mean and no more transitions are possible.

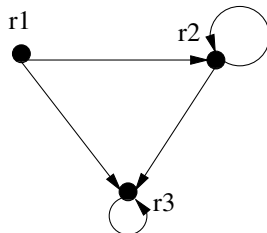


Fig. 1. Dependency graph for mean

This program has 2 components $\{r2\}$ and $\{r3\}$. Rule $r1$ does not belong to any component. Only constraints from the query match the head of the first rule. Since the query is a finite conjunction of constraints, this rule can only be executed a finite number of times.

The only constraints matching the head of rule $r2$, are constraints from the query and the body of rules $r1$ and $r2$. Rule $r1$ is only executed a finite number of times and the query is a finite conjunction of constraints. Therefore, termination of component $\{r2\}$ implies termination of the program consisting of $r1$ and $r2$.

The query and the constraints from the body of all rules match the head of rule $r3$. If rule $r1$ and $r2$ are only executed a finite number of times, an infinite computation only occurs with constraints from component $\{r3\}$. Therefore, to prove termination of this CHR program, it is sufficient to prove termination for components $\{r2\}$ and $\{r3\}$. \square

Definition 20. Program P with query I , satisfies the **component based termination condition**, iff for every component, all queries in $Call(P, I)$ terminate. \square

The example shows us a possible proof. Only rules which belong to a component, can be executed infinitely many times. There is always at least one component without outgoing edges. As in the example, termination can be proven for one component at a time, until the whole program is proven to terminate.

5 Automating the termination proof

To prove termination of a CHR program with the RC for CHR, one has to find an appropriate abstract level mapping, such that the RC is satisfied w.r.t. some associated interargument relations. We use abstract linear level mappings from Definition 11. In the philosophy of the constraint based approach to termination

analysis, as described in [4], we introduce a symbolic form of the level mapping. For a CHR constraint like $fib(N, M)$, we use a level mapping:

$$|fib(N, M)|^\alpha = fib_0 + fib_1 \times ||N||^\alpha + fib_2 \times ||M||^\alpha$$

with fib_0, fib_1, fib_2 unknown natural numbers. As a norm, abstract term-size or abstract list-length is used. Figure 2 shows the components of our prototype analyzer.

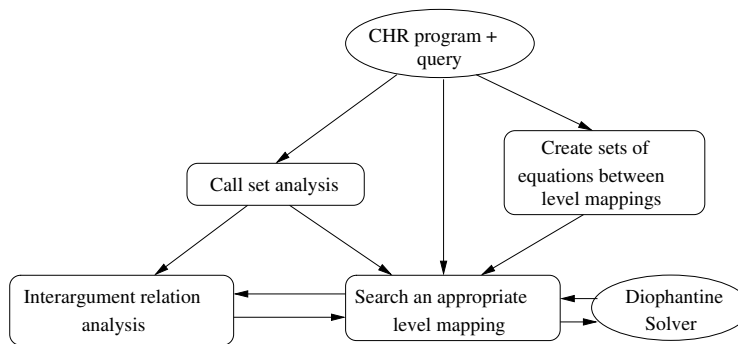


Fig. 2. Prototype analyzer

Call set analysis. This component performs a simple type inference to compute an overestimation of the call set. It is based on three basic types: *nil-terminated list*, *ground term* and *any term*. These types correspond to the norms abstract term-size and abstract list-length from Definition 10. We initialize the call set with the given query. 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 CHR body constraints are added to the call set.

Interargument relation analysis. Given a norm, abstract list-length or abstract term-size, and the information from the call set, interargument relations are computed. Built-in constraints manipulating lists or numbers, written in their symbolic notation, are used to find valid interargument relations w.r.t. the corresponding norm.

Create sets of equations between level mappings. This component will generate disjunctions of sets of equations representing the different possibilities for satisfying the RC for CHR in a program. For simplification or simpagation rules, these sets of equations correspond to different \succeq -maximal subsets from Definition 16 and can be described by sets of equations between level mappings. For a propagation rule there is only one set of equations.

Search an appropriate level mapping. This component searches a level mapping, for which every rule satisfies a set of equations. The equations between level mappings are transformed into Diophantine equations, using the method described in [10], which are then solved by the constraint solver of Aprove [9].

Component based termination condition. We implemented our condition of Definition 20. A CHR program P is split into components P_1, \dots, P_n , the call set analyzer is used to create the query for these components.

We demonstrate this by revisiting the Fibonacci example.

Example 13 (Automatic termination for Fibonacci). The analysis is performed on the Fibonacci program, given a query $fib(N, M)$ with N a *ground term* and M *any term*. The call set is initialized with the query. Because we ignore the number of constraints in the call set, the query matches every rules' head. The CHR body constraints are added to the constraint store. The call set remains the same, so we found a fix point: $\{fib(N, M) \mid N \text{ ground, } M \text{ any term}\}$.

In a next phase, we create the sets of equations between the level mappings of head and body constraints. The first rule, a simplification rule, has four possible ways of satisfying the RC for CHR. One of these possibilities needs to be satisfied by the selected level mapping and each of them corresponds to different \succeq -maximal subsets from Definition 16.

\succeq -maximal subset	Resulting set of equations
$\{fib(N, M1)\}$	$\{ fib(N, M1) ^\alpha \succ fib(N, M1) ^\alpha\}$
$\{fib(N, M2)\}$	$\{ fib(N, M2) ^\alpha \succ fib(N, M1) ^\alpha\}$
$\{fib(N, M1), fib(N, M2), fib(N, M1)\}$	$\{ fib(N, M1) ^\alpha \approx fib(N, M2) ^\alpha\}$
$\{fib(N, M1), fib(N, M2)\}$	$\{ fib(N, M1) ^\alpha \approx fib(N, M2) ^\alpha\}$

The set of equations for the second and third rule is \emptyset since they trivially satisfy the ranking condition. The last rule is a propagation rule and has only one possible way to satisfy the RC:

$$\begin{aligned} |fib(s(s(N)), M)|^\alpha &\succ |fib(s(N), M1)|^\alpha \\ |fib(s(s(N)), M)|^\alpha &\succ |fib(N, M2)|^\alpha \end{aligned}$$

The next component searches an appropriate level mapping. Abstract term-size is chosen as a norm and interargument relations are inferred. As there are no guards, no interargument relations are computed.

As a level mapping, $|fib(N, M)|^\alpha = fib_0 + fib_1 \times ||N||_t^\alpha + fib_2 \times ||M||_t^\alpha$ is used. This component searches natural numbers fib_0, fib_1 and fib_2 such that the level mapping proves termination for the program and query. We know fib_2 is 0, because the call set must be rigid w.r.t. the level mapping. The possible sets of equations between level mappings are combined and transformed into Diophantine constraints. Aprove finds the abstract level mapping given below, which proves termination for the Fibonacci program and the given query.

$$|fib(N, M)|^\alpha = 2 \times ||N||_t^\alpha$$

□

5.1 Experimental evaluation

We have tested our RC for CHR on a benchmark of 58 CHR programs, with CHR programs from [12] and WebCHR¹, and identified two problem classes: arc consistency algorithms and programs depending on matching. The results are given in the table below, all programs are terminating.

Ranking condition proves termination	39
Termination proven with the implementation	35
Termination proven with a non-standard level mapping	4
Failed proof	19
Failing Arc consistency algorithms	11
Failing because of matching	5
Failing other problem	3

The RC for CHR proves termination for two out of three programs from the benchmark. Only four programs which are proven to terminate with the RC for CHR, need a more complex level mapping than an abstract linear level mapping with abstract term-size or abstract list-length as a norm. So, these programs have been analyzed manually.

Arc consistency algorithms. Path and arc consistency algorithms on incomplete networks are programs containing a variant of the rule:

$$p(A, B, W1), p(B, C, W2) ==> \text{composition}(W1, W2, W) \mid p(A, C, W).$$

For this class of programs termination depends either on the query or on rule order. Current approaches fail to prove termination for this type of programs.

Matching. As stated, our technique fails to prove termination when termination depends on the evaluation of built-in constraints in the body of a rule. For some CHR programs, the CHR constraints in the body of the rule do not match any constraint from the head of a rule, unless a built-in from the body of the rule is evaluated. For these programs, this built-in constraint from the body should be taken into account.

6 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 these programs in standard CHR benchmarks.

¹ <http://chr.informatik.uni-ulm.de/~webchr/>

The condition on simplification rules, as proposed in [8], 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 approach of [8] succeeds.

To improve efficiency and usefulness of our ranking condition, a termination condition based on components has been examined and experimented with. The results are promising.

References

1. K. R. Apt and D. Pedreschi. Modular termination proofs for logic and pure prolog programs. In *116*, page 35. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, 31 1993.
2. Michael Codish and Samir Genaim. Proving termination one loop at a time. In *The 13th Workshop on Logic Programming Environments*, December 2003.
3. D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *Journal of Logic Programming*, 19-20:199–260, 1994.
4. S. Decorte, D. De Schreye, and H. Vandecasteele. Constraint based termination analysis of logic programs. *ACM Transactions on Programming Languages and Systems*, 21(6):1137–1195, 1999.
5. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1-2):69–116, 1987.
6. Gregory J. Duck, Peter J. Stuckey, María García de la Banda, and Christian Holzbaaur. The refined operational semantics of Constraint Handling Rules. In *20th International Conference on Logic Programming (ICLP'04)*, volume 3132 of *LNCS*, pages 90–104. Springer-Verlag, 2004.
7. T. Frühwirth. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1–3):95–138, October 1998.
8. T. Frühwirth. Proving termination of constraint solver programs. In *New Trends in Constraints*, volume 1865 of *LNCS*, pages 298–317. Springer-Verlag, 2000.
9. J. Giesl, P. Schneider-Kamp, and R. Thiemann. Aprove 1.2: Automatic termination proofs in the dependency pair framework. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, pages 281–286, 2006.
10. M. T. Nguyen and D. De Schreye. Polytool: Proving termination automatically based on polynomial interpretations. Technical report, Department of Computer Science, K.U.Leuven, Belgium, 2006.
11. M.-T. Nguyen, P. Schneider-Kamp, D. De Schreye, and J. Giesl. Termination analysis of logic programs based on dependency graphs. Report CW 496, K.U.Leuven, Department of Computer Science, June 2007.
12. P. Pilozzi, T. Schrijvers, and D. De Schreye. Proving termination of CHR in Prolog: a transformational approach. In *9th International Workshop on Termination*, 2007.
13. T. Schrijvers. *Analyses, Optimizations and Extensions of Constraint Handling Rules*. PhD thesis, K.U.Leuven, Leuven, Belgium, June 2005.
14. T. Schrijvers and T. Frühwirth. Optimal Union-Find in Constraint Handling Rules. *Theory and Practice of Logic Programming*, 6(1&2), 2006.
15. J. Sneyers, T. Schrijvers, and B. Demoen. The computational power and complexity of Constraint Handling Rules. In *2nd Workshop on Constraint Handling Rules (CHR'05)*, pages 3–17, October 2005.