Proving termination of CHR in Prolog: A transformational approach

Paolo Pilozzi
Tom Schrijvers
Danny De Schreye

Report CW 487, April 2007



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Proving termination of CHR in Prolog: A transformational approach

Paolo Pilozzi
Tom Schrijvers
Danny De Schreye

Report CW 487, April 2007

Department of Computer Science, K.U.Leuven

Abstract

In this paper we present a termination preserving transformation from Constraint Handling Rules to Prolog. The transformation is sound w.r.t. termination under the theoretical semantics of Constraint Handling Rules. It does not consider the presence of a propagation history. The transformation allows for the direct reuse of termination proof methods from Logic Programs and Term-Rewrite Systems, yielding the first fully automatic termination proving for Constraint Handling Rules. We formalize the transformation and show usefulness of the approach. We transform a set of CHR programs, by an implementation of the transformation and show termination by using existing termination tools for Logic Programs and Term-Rewrite Systems.

Keywords : Constraint Handling Rules, Termination Analysis, Program Transformation.
CR Subject Classification : D.3.2 Language classifications - Constraint and logic languages, F.3.2 Semantics of Programming Languages - Program analysis, I.2.2 Automatic Programming - Program transformation

Proving termination of CHR in Prolog: A transformational approach

Paolo Pilozzi*, Tom Schrijvers**, and Danny De Schreye

Department of Computer Science, K.U.Leuven, Belgium
{Paolo.Pilozzi, Tom.Schrijvers, Danny.DeSchreye }@cs.kuleuven.be

Abstract. In this paper we present a termination preserving transformation from Constraint Handling Rules to Prolog. The transformation is sound w.r.t. termination under the theoretical semantics of Constraint Handling Rules. It does not consider the presence of a propagation history. The transformation allows for the direct reuse of termination proof methods from Logic Programs and Term-Rewrite Systems, yielding the first fully automatic termination proving for Constraint Handling Rules. We formalize the transformation and show usefulness of the approach. We transform a set of CHR programs, by an implementation of the transformation and show termination by using existing termination tools for Logic Programs and Term-Rewrite Systems.
Keywords: Constraint Handling Rules, Termination Analysis, Program Transformation

1 Introduction

Constraint Handling Rules (CHR), invented by Thom Frühwirth [5], is a relatively young member of the family of declarative programming languages. It is a concurrent, comitted-choice, logic programming language. CHR is constraint-based and has guarded rules that rewrite multisets of atomic formulas. Its simple syntax and semantics make it well-suited for implementing custom constraint solvers [5, 12]. It is the latter feature of the language that caused its success and impact on the research community. Many solvers were implemented in CHR, from consistency techniques, over temporal constraint solvers, to solvers for web-ontologies [7]. Despite the amount of work directed to CHR [12, 4, 14, 15], not much work has been done on termination analysis and to the best of our knowledge, no attempts were made to automate termination proofs.

The first and, until now, only contribution to termination analysis of CHR is reported in [6] and shows termination of CHR constraint solvers under the theoretical semantics of CHR. Termination is shown, using a *ranking function*, mapping sets of constraints to a well-founded order. A *ranking condition*, on the

* Supported by the Fund for Scientific Research - Flanders (Belgium) (F.W.O. - Vlaanderen): "Termination analysis, crossing paradigm borders".
** Post-Doctoral Researcher of the Fund for Scientific Research - Flanders (Belgium) (F.W.O. - Vlaanderen).

level of the rules, implies termination. It is the direct counterpart of the *norm* and *level mappings* used in Logic Programs (LP). One difference, however, is that in LP it suffices to reason about the rank or level values of individual atoms. If the natural number ranking associated to atoms decreases from any call to any recursive call, we have a termination proof. In CHR we need a different argumentation because CHR constraints are simultaneously replaced by a number of new constraints, or - for propagation rules - introduce new constraints. Although termination conditions in CHR take a different form than in LP and Term-Rewrite Systems (TRS), [6] shows that achievements from the work on termination of LP [2] and TRS [3] are relevant and adaptable to the CHR context.

In this paper, we present a termination preserving transformation of CHR to Prolog. This allows the direct reuse of termination proof methods from LP and TRS for CHR, yielding the first fully automatic termination proving for CHR. The transformation is termination preserving under the theoretical semantics of CHR and does not take a propagation history into account. Therefore, we cannot prove termination of CHR programs with propagation rules. We formalize the transformation and prove soundness w.r.t. termination. We implemented the transformation and used existing termination tools for LP and TRS on a set of CHR programs to demonstrate the usefulness of our approach. This paper is organized as follows:

Section 2 introduces CHR by its syntax and its declarative and operational semantics. We shortly introduce the in-language approach to termination analysis presented in [6].
Section 3 discusses the different aspects of the executional behaviour of CHR programs and presents a practical termination preserving transformation of CHR programs to Prolog programs.
Section 4 presents a formalization of the transformation to establish soundness of the transformation w.r.t. termination.
Section 5 demonstrates the usefulness of the approach by results on termination of a set of CHR programs. Termination is shown due to an implementation of the transformation and termination tools for LP and TRS.
Section 6 concludes this text and discusses future work.

2 Introduction to CHR

In this section we discuss syntax, declarative semantics and simple operational semantics of CHR. This section serves mainly as an introduction to the language of CHR. For a more elaborate introduction we refer to [1]. At the end of this section, we introduce the in-language approach presented in [6].

2.1 CHR Syntax

CHR is a programming language for the development of custom constraint solvers. This requires a syntactical representation of constraints, such that rules can relate sets of constraints with other sets of constraints. In CHR, a constraint is considered to be a distinguished, first-order predicate and is defined as follows.

Definition 1 (Constraint). *In CHR, a constraint is represented as a predicate, $c(t_1, t_2, ..., t_n)$, of arity n, with $n \geq 0$. There are two distinct kinds of constraints in CHR, i.e. built-in constraints and CHR constraints. Built-ins are defined in the host-language and solved by the underlying constraint solver. CHR constraints are defined in the CHR program and solved by CHR rules.*

A CHR program can implement three kinds of rules, a simplification rule which simplifies constraints in simpler constraints, a propagation rule, which adds redundant information that could allow for further simplifications and a simpagation rule, which is a rule that simplifies constraints conditional on the presence of other constraints. These rules are syntactically defined as follows.

Definition 2 (Syntax of a CHR Program). *A CHR program P is a finite set of CHRs. There are three kinds of CHR:*

A simplification CHR:
$$[\ N\ @\ ]\ H \Leftrightarrow [\ G\ |\ ]\ B.$$
A propagation CHR:
$$[\ N\ @\ ]\ H \Rightarrow [\ G\ |\ ]\ B.$$
A simpagation CHR:
$$[\ N\ @\ ]\ H_1 \setminus H_2 \Leftrightarrow [\ G\ |\ ]\ B.$$

Square brackets denote optional elements, i.e. an optional name and an optional guard. The multi-head H (or H_1 and H_2) of a rule represents a conjunction of head constraints and can only contain CHR constraints. The body B, is a conjunction of built-in constraints and CHR constraints. The optional guard G is a conjunction of built-in constraints.

As in Prolog syntax, a conjunction of constraints is written as a sequence of conjuncts separated by commas. Empty sequences are denoted by the built-in constraint *true*. Note that by having an empty left or right head in a simpagation rule, we get respectively a simplification and a propagation rule.

We generalize CHR rules by simpagation rules and all futher definitions are therefore only given for simpagation rules. For explanatory purposes we still differentiate between the three kinds of rules.

2.2 CHR Declarative Semantics

The declarative semantics of a CHR program P, is a conjunction of universally quantified formulas (one for each rule) and a consistent built-in constraint theory, CT, which determines the meaning of the built-in constraints appearing in P. The theory CT is expected to include a syntactical equality constraint $=$ and the basic trivial constraints *true* and *false*.

The declarative reading of a rule, relates heads and bodies, provided that the guards are true.

Definition 3 (Declarative semantics of CHR). *Declaratively,*

- *a CHR rule is a logical implication of a logical equivalence, given that the guard is satisfied:*
$$\forall(G \rightarrow (H_1 \rightarrow (H_2 \leftrightarrow \exists \bar{z}(B))$$

A CHR rule relates constraints on the presence of other constraints. When substituting respectively the left and right head of a CHR rule by the constraint *true* we get the declarative meaning of a simplification and propagation rule.

A simplification rule means that the heads are equivalent to the body given that the guard is true. A propagation rule means that the heads imply the body given that the guard is true. A simpagation rule is a hybrid kind of rule, as the presence of constraints (left head) imply an equivalence between constraints (right heads and bodies), conditional on the guard.

2.3 CHR Operational Semantics

CHR constraint solvers are programmed by defining rules. Declaratively, rules define equivalent states of constraint stores. Operationaly, these stores are derived one after another.

The operational semantics of CHR programs is given by a state transition system. These transitions are also called computation steps where one can proceed from one state to the next. A sequence of these computation steps is called a computation.

Definition 4 (CHR execution state). *A* CHR execution state *or* CHR goal *is a conjunction of built-in and CHR constraints. This is often referred to as a* constraint store *and both store and state are used interchangeably. An* initial state *or* query *is an arbitrary goal state. In a* final state *or* answer *either the built-in constraints are inconsistent, causing finite failure, or no more computation steps are possible resulting in an answer for the query.*

A CHR program defines transitions between CHR execution states based on the rules in the program. Let P be a CHR program for the CHR constraints and CT be a constraint theory for the built-in constraints, then the *transition relation* $\longmapsto$ between states is defined as follows.

Definition 5 (CHR Transition Relation).

$$H'_1 \wedge H'_2 \wedge D \longmapsto (H_1 \wedge H_2 = H'_1 \wedge H'_2) \wedge G \wedge B \wedge H'_1 \wedge D$$
$$\textit{if } (H_1 \setminus H_2 \Leftrightarrow G \mid B) \textit{ in } P \textit{ and } CT \models D \rightarrow \exists \bar{x}((H_1 \wedge H_2 = H'_1 \wedge H'_2) \wedge G)$$

The variables occurring in the rule chosen from P, are denoted by $\bar{x}$*. By equating two atomic constraints,* $c(t_1, ..., t_n) = c(s_1, ..., s_n)$ *syntactically, we mean* $(t_1 = s_1) \wedge ... \wedge (t_n = s_n)$*. By* $(H_1 \wedge ... \wedge H_n) = (H'_1 \wedge ... \wedge H'_n)$ *we mean* $(H_1 = H'_1) \wedge ... \wedge (H_n = H'_n)$*. Conjuncts can be permuted since conjunction is assumed to be associative and commutative.*

When using a rule from the program, its variables are renamed using new symbols. A rule is applicable to CHR constraints, H', whenever these atomic constraints match (are instance of) the head atoms, H, of the rule and when the guard, G, is entailed (implied) by the built-ins in the constraint store. The matching is the effect of the existential quantification in $\exists \bar{x}(H = H')$. Matching and entailment checks are performed by the constraint solver for the built-in

constraints. Any of the applicable rules can be applied, however the choice of which rule is applied is a commited choice, it cannot be undone.

As propagation rules do not remove constraints from the store, they can fire the propagation rule again. This is called trivial non-termination. A propagation history can prevent this from happening by allowing a propagation rule to fire at most once on the same set of constraints. In the remainder of this text, we do not consider programs that require a propagation history to terminate.

The program below implements merge-sort[1]. The query $mergesort(L)$ with L a list of length 2^n, yields a tree-representation of the order.

Example 1. Mergesort

$mergesort([]) \Leftrightarrow true.$
$mergesort([L|List]) \Leftrightarrow root(0, L), mergesort(List).$
$root(D, L1), root(D, L2) \Leftrightarrow less(L1, L2) \mid root(s(D), L1), edge(L1, L2).$

The first two rules decompose a list of elements, while adding new *root/2* constraints to the store. The constraint $root(D, L)$ represents a tree of depth D (initially 0) and root value L. The third rule performs the actual merge-sorting by joining two trees of equal depth. It replaces both trees by a new tree of increased depth, where the largest root becomes a child node of the smallest.

2.4 Termination analysis for CHR

Termination of CHR programs can be shown by mapping CHR states to a well-founded order, such that a decrease is shown between all consecutive states in the CHR program. As a CHR state is a conjunction of constraints, which is commutable and associative, we require a mapping that is invariant over permutations of the constraint store.

Two such mappings are used in [6]. Both approaches prove termination by relying on polynomial interpretations, where the rank of a term or atom is defined by a linear positive combination of the rankings of its arguments. Built-in constraints imply interargument relations between heads and bodies and are themselves mapped to 0, as they are assumed to terminate on their own. The ranking value of a variable is defined to be greater or equal to zero.

Sets of constraints are either compared by using a commutable and associative operator on the ranking values of the constraints in the set or by comparing them by a multiset order. In the latter case we compare sets by ordering the ranking values of the constraints and comparing both sets lexicographically.

Conditions on the level of rules imply termination of the CHR program. That is, by showing for all rules in the program, a decrease between the set of constraints that is removed from the store and the set that is added to the store.

In case of propagation rules, there are no CHR constraints removed from the constraint store. Therefore, for programs with propagation rules, it is impossible to show a decrease between consecutive states where the transition is due to a propagation rule. To show termination on these kind of programs we need to consider a propagation history.

[1] Based on an example from `http://www.cs.kuleuven.ac.be/~dtai/projects/CHR/`

3 A transformational Approach to Termination Analysis of CHR Programs

In this section we will introduce a transformation of a CHR program to a termination equivalent Prolog program. We will show that such a transformation exists, that it can be done in a straightforward way and that it can be done without obfuscating the termination argument. We first address non-determinism in CHR caused by the fair selection of rules principle. Afterwards we discuss a representation in Prolog for the CHR constraint store and we will show how to transform CHR rules to Prolog clauses. Finally, we will discuss how to represent the operational semantics of CHR programs in Prolog.

3.1 About non-determinism of CHR programs

The source of non-determinism in CHR programs is the fair selection of rules principle and has two different causes. The first cause is the same set of constraints that can fire multiple rules. This is shown in the next example.

Example 2. Same set of constraints can fire multiple rules

$rule_1$ @ $a, a \Leftrightarrow a$.
$rule_2$ @ $a, a \Leftrightarrow a, a, a$.

The CHR program above, may or may not terminate for an initial goal store $\{a, a\}$. It depends on which rules are fired, as both rules fire on the same set of constraints. We can, therefore, never be sure which one of these rules will fire in the execution of the program.

A second cause is that different combinations of constraints could fire different rules in the CHR program. This is illustrated below.

Example 3. Different sets of constraints can fire a rule

$rule_1$ @ $a \Leftrightarrow c$.
$rule_2$ @ $b \Leftrightarrow d$.
$rule_3$ @ $a, d \Leftrightarrow a, b$.

The CHR program above, may or may not terminate for a given initial store $\{a, b\}$. The cause of this non-determinism is, however, of a different kind. It depends on which constraints are selected from the store. If it is the first rule that fires, then the program terminates immediately, if it are the second and the third rule which fire repeatedly, then the program runs forever.

We can represent the non-determinism of CHR by search in Prolog. This is achieved by representing CHR rules as clauses with the same head predicate.

3.2 Representing the CHR Constraint Store in Prolog

The CHR constraint store is a commutable conjunction of constraints. In Prolog we represent the store as a list of constraints and call them storelists. These lists

enumerate the constraints in the store and are unified unorderly by unification of their permutations. For each store with n elements there exists therefore a set, Q, of $n!$ storelists, that are all equivalent enumerations of the constraint store.

$$S = constr_1 \wedge constr_2 \wedge ... \wedge constr_n \leftrightarrow \\ \{L \mid permutation([constr_1, constr_2, ..., constr_n], L)\} = Q$$

Here, $permutation/2$ is defined to be true whenever the arguments are list of the same length, from which the same elements can be removed.

3.3 Representing CHR Rules in Prolog

A CHR rule transforms a CHR constraint store in another CHR constraint store if the guard, given the matching constraints, is satisfied. CHR rules are committed choice, but this notion can be left unconsidered as we require all computations to be finite.

We represent the non-determinism of CHR programs as search in Prolog. We obtain such a behaviour by transforming every CHR rule to a Prolog clause with as a head predicate: $rule/2$. This generalizes the way how a rule is fired with a particular storelist and represents the non-determinism caused by the fair selection of rules principle.

Each rule clause in Prolog defines consecutive stores, defined by the original CHR rule. This is a relation between storelists in Prolog. A call to $rule/2$ with a list representation of the store results in a unification of the permutation of the current store with a storelist representation of the rule's head. If such a unification exists such that all body atoms succeed, then the second term of $rule/2$ will bind to a list enumerating the next store. I.e. a CHR rule of the form:

$$H_1, ..., H_i \setminus H_j, ..., H_n \Leftrightarrow G_1, ..., G_k \mid B_{BI_1}, ..., B_{BI_l}, B_{CHR_1}, ..., B_{CHR_m}.$$

becomes a Prolog clause:

$$\begin{aligned} rule(S, [H_1, ..., H_i, B_{CHR_1}, ..., B_{CHR_m} | R]) \ :- \\ & permutation(S, [H_1, ..., H_n | R]), \\ & G_1, ..., G_k, \\ & B_{BI_1}, ..., B_{BI_l}. \end{aligned}$$

Note that the head of the rule is represented as a list with a variable as tail. The call to permutation generalizes the matching process between the head and the store, by finding a permutation of the store that unifies with the list representation of the head. When such a unifier is found it will bind the tail, R, to the remainder of the store. To obtain the new store, we add to the remainder of the the old store, the left heads of the simpagation rule together with the CHR body constraints appearing in the rule.

3.4 Representing the operational semantics of CHR in Prolog

The operational semantics of CHR programs is already largly represented by the rule clauses. Matching of constraints in the store with heads of the rules is done by finding a permutation of the store which unifies with the heads of that rule. The resulting store is contained in the second argument of the rule clause. We only have to call rules repeatedly.

$$
\begin{array}{l}
rule([H_1, ..., H_n|R], [H_1, ..., H_i, B_{CHR_1}, ..., B_{CHR_m}|R]) \text{ :-} \\
\qquad G_1, ..., G_k, B_{BI_1}, ..., B_{BI_l}. \\
goal(S) \text{ :-} \\
\qquad permutation(S, PS), \\
\qquad rule(PS, NS), \\
\qquad goal(NS). \\
goal(_).
\end{array}
$$

Note that because the permutation is present in every rule it can be factored out to the goal clause. Also note that whenever the program cannot call any of the rule clauses, it will end up in a refutation due to $goal(_)$. This represents termination in CHR. Note that due to $goal(_)$ we introduce at each call to $goal/1$ a path ending in a refutation.

Example 4. Mergesort cont. We revisit *mergesort* and transform each of the rules into their clausal form. We assume that *less/2* is predefined.

$$
\begin{array}{l}
rule([mergesort([])|R], R). \\
rule([mergesort([L|List])|R], [root(0, L), mergesort(List)|R]). \\
rule([root(D, L1), root(D, L2)|R], [root(s(D), L1), edge(L1, L2)|R]) \text{ :-} \\
\qquad less(L1, L2).
\end{array}
$$

$$
\begin{array}{l}
goal(S) \text{ :-} \\
\qquad permutation(S, PS), \\
\qquad rule(PS, NS), \\
\qquad goal(NS). \\
goal(_).
\end{array}
$$

A query for this transformed program is of the form $goal([mergesort(L)])$.

3.5 Termination of the transformed CHR programs

Termination proofs of the transformed program take a general form, showing a decrease in level mapping: $|goal(S)| > |goal(NS)|$. This condition boils down to a condition on the norms of the arguments of all *rule/2* clauses: $\| PS \|_{sl} > \| NS \|_{sl}$. Here $\| \, . \, \|_{sl}$ denotes the norm[2] used on storelists.

[2] The norm should be invariant over permutation.

Example 5. Mergesort cont. The following conditions imply termination of *mergesort*:

$$\begin{aligned} \| \, [mergesort([])|R] \, \|_{sl} &> \| \, R \, \|_{sl} \\ \| \, [mergesort([L|List])|R] \, \|_{sl} &> \| \, [root(0, L), mergesort(List)|R] \, \|_{sl} \\ \| \, [root(D, L1), root(D, L2)|R] \, \|_{sl} &> \| \, [root(s(D), L1), edge(L1, L2)|R] \, \|_{sl} \end{aligned}$$

These conditions are met by the following storelist norm, $\| \cdot \|_{sl}$:
$\| \, [E_1, ..., E_n] \, \|_{sl} = \sum_1^n \| \, E_1 \, \|_c$. Here, $\| \, . \, \|_c$ are constraint norms:

$$\begin{aligned} \| \, mergesort(List) \, \|_c &= 1 + 2 \, \| \, List \, \|_l \\ \| \, root(D, L) \, \|_c &= 1 \\ \| \, edge(A, B) \, \|_c &= 0 \end{aligned}$$

and $\| \cdot \|_l$ is the usual list-length norm.

4 Formalization of the transformation

In this section we formalize the transformation and establish soundness w.r.t. termination of CHR programs. We define the transformation and define computations and derivations for respectively CHR and LP. Afterwards, we redefine derivations as execution steps, based on the invariant part of the transformation.

Definition 6 (Transformation). *A CHR program P is transformed into the following logic program $\wp$.*

- *The logic program $\wp$ contains following clauses:*
  $goal(S)$:- $perm(S, PS)$, $rule(PS, NS)$, $goal(NS)$.
  $goal(_)$.
  $perm(L, [X|P])$:- $del(X, L, L1)$, $perm(L1, P)$.
  $perm([], [])$.
  $del(X, [Y|T], [Y|R])$:- $del(X, T, R)$.
  $del(X, [X|T], T)$.
- *The logic program $\wp$ contains for every rule:*
  $H_1, ..., H_i \setminus H_j, ..., H_n <=>$
  $G_1, ..., G_k \mid$
  $B_{B_1}, ..., B_{B_l}$,
  $B_{C_1}, ..., B_{C_m}$.
  in P, a clause:
  $rule([H_1, ..., H_n|T], [H_1, ..., H_i, B_{C_1}, ..., B_{C_m}|T])$:-
  $G_1, ..., G_k$,
  $B_{B_1}, ..., B_{B_l}$.

The program independent part calls a rule with a storelist, which, whenever the call succeeds, binds the second argument of $rule/2$ to the next list representation of the store.

To show that the transformed program is termination preserving we have to define the operational semantics of CHR and LP. Although, we gave a definition of the operational semantics before, we will redefine it here in the context of computations rather than transitions.

Definition 7 (Computation step, Computation, Termination). *For a CHR program P and a constraint theory CT, we define:*

Let $S = H'_1 \wedge H'_2 \wedge D$ be a query and let r @ $H_1 \setminus H_2 <=> G \mid B$. be a CHR rule in P. Then S' is a resolvent of S and r using a variable assignment $\bar{x}$, such that $S \vdash_{r,\bar{x}} S'$ if $CT \models D \rightarrow \exists\bar{x}(((H_1 \wedge H_2) = (H'_1 \wedge H'_2)) \wedge G)$. $S \vdash_{r,\bar{x}} S'$ is called a computation step.
A possibly infinite sequence $S_0, S_1, ...$ is called a computation if for all S_i and S_{i+1} holds that $S_i \vdash_{r_{i+1},\bar{x}_{i+1}} S_{i+1}$, where r_{i+1} is a fresh variant of a rule r in a CHR program P and where $\bar{x}_{i+1}$ is a variable assignment.
A program P terminates if all its computations starting from a query $S = S_0$ are finite.

In a similar way derivations and derivation steps are defined for LP.

Definition 8 (Derivation step, Derivation, Termination). *For a logic program π we define:*

Let $Q =\leftarrow A_1, ...A_n.$ be a query and let $c = H :\text{-} B_1, ..., B_m.$ be a clause in π. Then Q' is a resolvent of Q and c using θ, such that $Q \vdash_{c,\theta} Q'$ if $\theta = mgu(A1, H)$ and $Q' =\leftarrow B_1, ..., B_m, A_2, ..., A_n.$ $Q \vdash_{c,\theta} Q'$ is called a derivation step.
A possibly infinite sequence $Q_0, Q_1, ...$ is called a derivation if for all Q_i and Q_{i+1} holds that $Q_i \vdash_{c_{i+1},\theta_{i+1}} Q_{i+1}$, where c_{i+1} is a fresh variant of a clause c in a logic program π and where θ_{i+1} is a substitution.
A program π terminates if all its derivations starting from a query $Q = Q_0$ are finite.

To show similarity between the execution of the transformed program and the original CHR program, we redefine derivations under LP in a program specific way. This is achieved by the concept execution path where we take into account the program independent part of the transformation.

Definition 9 (Execution step, Execution path, Termination). *For a logic program $\wp$, we redefine derivation steps as execution steps in following way:*

Let $Q =\leftarrow goal([C_1, ..., C_n])$ be a query and let $\wp$ be a finite set of clauses as defined in Definition 6. Then $Q' =\leftarrow goal([C'_1, ..., C'_n])$. is a resolvent of Q and $\wp$ using θ, such that if $\wp \models rule([C_1, ..., C_n], [C'_1, ..., C'_n])$ then $Q \vdash_{\wp,\theta} Q'$ where $\theta = \sigma_1 ... \sigma_n$. $Q \vdash_{\wp,\theta} Q'$ is called an execution step.
A possibly infinite sequence $Q_0, Q_1, ...$ is called an execution path if for all Q_i and Q_{i+1} holds that $Q_i \vdash_{\wp,\theta_{i+1}} Q_{i+1}$ where θ_{i+1} is a substitution and $\wp$ is a program as in Definition 6.
A program $\wp$ terminates if all its execution paths starting from a query $Q = Q_0$ are finite.

We now show that derivations are captured by execution paths given the invariant part of the transformation. For all queries $goal/1$ there exist a finite

derivation to a refutation or to another query of the type $goal/1$. These derivations correspond to execution steps. We show that a derivation to $goal/1$ is only possible if a call to $rule/2$ succeeds. We also show that there exist between execution steps no other call to $goal/2$.

Theorem 1. *Given a query $Q =\leftarrow goal([C_1, ..., C_n])$ and a logic program $\wp$, then there is either:*

a finite derivation to query $Q' =\leftarrow goal([C'_1, ..., C'_n])$,
such that $\wp \models rule([C_1, ..., C_n], [C'_1, ..., C'_n])$, or either
a finite derivation to a refutation, $\square$, due to $goal(_)$.

Theorem 2. *For two queries Q_i and Q_j, such that $Q_i \vdash_{\wp,\theta_j} Q_j$ is an execution step, then there does not exist a query of the form $Q_{i+k} =\leftarrow goal([C_1, ..., C_n])$ in its corresponding derivation $Q_i, ..., Q_{i+k}, ..., Q_{i+n} = Q_j$.*

By connecting both the transformed program and the original program we show that all computations in CHR are represented in LP by execution paths and that all execution paths are represented by computations in CHR. This almost trivially implies that termination is preserved in the transformed program.

The following definitions are required to connect both the original CHR program and the transformed LP. We define the domain of both programs as follows.

Definition 10 (Domain of P and $\wp$). *The domain of P is defined as the set of queries, $S = C_1 \wedge ... \wedge C_n$ and is denoted as Σ_P. The domain of $\wp$ is defined as the set of queries, $Q =\leftarrow goal([C_1, ..., C_n])$, and is denoted as $\Sigma_\wp$. The constraints C_i are defined as usual.*

The following semantic functions map respectively stores to next stores and storelists to next storelists. We will use them to establish equivalence between the execution of CHR programs and that of the transformed programs.

Definition 11 (Semantic function f_P). *The semantic function $f_P : \Sigma_P \rightarrow \Sigma_P$, is defined as $f_P(\{S_1\}) = \{S_2 \mid S_1 \vdash_{P,\bar{x}} S_2\}$.*

Definition 12 (Semantic function $f_\wp$). *The semantic function $f_\wp : \Sigma_\wp \rightarrow \Sigma_\wp$, is defined as $f_\wp(\{Q_1\}) = \{Q_2 \mid Q_1 \vdash_{\wp,\theta} Q_2\}$.*

We define two functions that represent respectively a mapping of storelists to stores one of stores to sets of storelists.

Definition 13 (α). *The function α is defined as a function mapping queries of $\wp$ on queries of P: $\alpha(\{\leftarrow goal([C_1, C_2, ..., C_n])\}) = \{C_1 \wedge C_2 \wedge ... \wedge C_n\}$.*

Definition 14 (γ). *The function γ is defined as α^{-1}, in other words:*
$\gamma(\{C_1 \wedge C_2 \wedge ... \wedge C_n\}) = \{\leftarrow goal([C_{i+1}, C_{i+2}, ..., C_{i+n}]) \mid$
$\wp \models perm([C_1, C_2, ..., C_n], [C_{i+1}, C_{i+2}, ..., C_{i+n}])\}$

Now it is time to connect both $\wp$ and P such that we can show that the transformed program represents all computations of the original CHR program and therefore represents termination.

Theorem 3 (Connecting $\wp$ and P). *There exist following relations between a transformed CHR program $\wp$ and the original CHR program P:*



$$\forall Q_1 : f_\wp(\{Q_1\}) \subseteq \gamma \circ f_P \circ \alpha(\{Q_1\}) \; \forall S_1 : f_P(\{S_1\}) = \alpha \circ f_\wp \circ \gamma(\{S_1\})$$

This final theorem establishes the equivalence of termination of the transformed program and the original programs.

Theorem 4 (Soundness of the Transformation). *Let P be a CHR program and let $\wp$ be its transformed logic program. Then termination of a query S given a CHR program P is equivalent to termination of all queries in $\gamma(S)$ given a logic program $\wp$.*

5 Evaluation

We have implemented the transformation[3] in K.U.Leuven CHR [13] for SWI-Prolog [17]. The implementation was used to transform two sets of CHR programs, one based on [9] benchmarks and another from [6] marked with a '*'. Polytool (PT) [11] and AProVE (AP) [8], based on respectively LP and TRS termination proof techniques, were used to prove termination of the transformed CHR programs:

	PT	AP		PT	AP
ackermann	-	-	convert	-	-
average	+	+	diff	+	+
binlog	+	+	factorial	+	+
booland*	+	+	gcd	+	+
boolcard*	-	+	genint	+	+
concat	-	-	joinlists	+	+
linpoleq*	-	+	pathcons*	-	+
max	+	+	power	+	+
mean	+	+	revlist	+	+
mergesort	+	+	som	+	+
modulo	+	+	toyama	+	+
oddeven	+	+	weight	-	-

Similar, but weaker results were obtained with the LP termination tools cTI [10] and TerminWeb [16]. The results show that the transformation conserves the termination argument, that analysis can be done using approaches from LP and TRS and that it can be automated in terms of existing termination tools.

[3] Available at `http://www.cs.kuleuven.be/~paolo/`

6 Conclusion

We have introduced a straightforward program transformation that allows for termination proofs of CHR programs, by using existing tools for LP and TRS. We have implemented the transformation and obtained good results, which confirm the applicability of proof techniques of LP and TRS to CHR. Termination proofs take a general form as was suggested by [6] and can be automated by using existing termination analysers for LP and TRS.

Future work will address CHR programs with rules that do not remove constraints from the store. These rules introduce trivial non-termination, prevented by a propagation history, which isn't supported by our current transformation.

References

1. Slim Abdennadher. Operational Semantics and Confluence of Constraint Propagation Rules. In Gert Smolka, editor, *CP'97: Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming*, pages 252–266, Schloss Hagenberg, Austria, 1997. Springer-Verlag.
2. D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *Journal of Logic Programming*, 19-20:199–260, 1994.
3. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1-2):69–116, 1987.
4. Gregory J. Duck, Peter J. Stuckey, María García de la Banda, and Christian Holzbaur. The refined operational semantics of Constraint Handling Rules. In Bart Demoen and Vladimir Lifschitz, editors, *20th International Conference on Logic Programming (ICLP'04)*, volume 3132 of *Lecture Notes in Computer Science*, pages 90–104, Saint-Malo, France, September 2004. Springer-Verlag.
5. T. Frühwirth. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1–3):95–138, October 1998.
6. T. Frühwirth. Proving termination of constraint solver programs. In Krzysztof R. Apt, Antonis C. Kakas, Eric Monfroy, and Francesca Rossi, editors, *New Trends in Constraints*, volume 1865 of *Lecture Notes in Computer Science*, pages 298–317, Paphos, Cyprus, 2000. Springer-Verlag.
7. Thom Frühwirth and Pascal Brisset. Optimal Placement of Base Stations in Wireless Indoor Telecommunication. In Michael J. Maher and Jean-Francois Puget, editors, *CP'98: Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, volume 1520 of *Lecture Notes in Computer Science*, pages 476–480, Pisa, Italy, October 1998. Springer-Verlag.
8. 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.
9. C. Marché and H. Zantema. The termination problems database 3.2, http://www.lri.fr/∼marche/tpdb/, viewed november 2006.
10. F. Mesnard and R. Bagnara. Cti: a constraint-based termination inference tool for iso-prolog. *Theory and Practice of Logic Programming*, 5(1-2):243–257, 2005.
11. M. T. Nguyen and D. De Schreye. Polynomial interpretations as a basis for termination analysis of logic programs. In G. Gupta M. Gabbrielli, editor, *Proceedings of the 21st International Conference on Logic Programming (ICLP'05)*, volume 3668 of *LNCS*, pages 311–325. Springer Verlag, 2005.

12. Tom Schrijvers. *Analyses, Optimizations and Extensions of Constraint Handling Rules*. PhD thesis, K.U.Leuven, Leuven, Belgium, June 2005.
13. Tom Schrijvers and Bart Demoen. The K.U.Leuven CHR system: Implementation and application. In Thom Frühwirth and Marc Meister, editors, *First Workshop on Constraint Handling Rules: Selected Contributions*, pages 1–5, Ulm, Germany, May 2004.
14. Tom Schrijvers and Thom Frühwirth. Optimal Union-Find in Constraint Handling Rules. *Theory and Practice of Logic Programming*, 6(1&2), 2006.
15. Jon Sneyers, Tom Schrijvers, and Bart Demoen. The computational power and complexity of Constraint Handling Rules. In T. Schrijvers and T. Frühwirth, editors, *2nd Workshop on Constraint Handling Rules (CHR'05)*, pages 3–17, Sitges, Spain, October 2005.
16. C. Taboch, S. Genaim, and M. Codish. Terminweb: Semantic based termination analyser for logic programs, http://www.cs.bgu.ac.il/∼mcodish/terminweb, 2002.
17. Jan Wielemaker. SWI-Prolog release 5.6.0, 2006. http://www.swi-prolog.org/.