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


1 Introduction

Constraint Handling Rules (CHR) is a concurrent, committed-choice, logic programming language. It is constraint-based and has guarded rules that rewrite multisets of atomic formulas [1]. Its simple syntax and semantics make it well-suited for implementing custom constraint solvers. Despite the amount of work directed to CHR, not much has been done on termination analysis. To the best of our knowledge, there were no attempts made to automate termination proofs.

The first and, until now, only contribution to termination analysis of CHR is reported in [2] and shows termination of CHR programs under the theoretical semantics [1] of CHR. Termination is shown, using a ranking function, mapping sets of constraints to a well-founded order. A ranking condition, on the level of the rules, implies termination. Although termination conditions in CHR take a different form than in Logic Programs (LP) and Term-Rewrite Systems (TRS), [2] shows that achievements from the work on termination of LP and TRS 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. 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. In [3], we formalize the transformation and prove soundness w.r.t. termination.

2 Syntax and Semantics of CHR

We briefly introduce the necessary aspects of CHR. For a more elaborate introduction see [1].

* 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).
Syntax. A constraint in CHR is a first-order predicate. We distinguish between built-in constraints, pre-defined and solved by the underlying constraint solver, and CHR constraints, user-defined and solved by a finite set of CHR rules (CHR program). Rules are of the form:

\[ H_1 \setminus H_2 \leftrightarrow G \mid B. \]

Both parts of the multihead, \( H_1 \) and \( H_2 \), are conjunctions of CHR constraints. The body, \( B \), is a conjunction of built-in and CHR constraints. The optional guard, \( G \), is a conjunction of built-in constraints. Empty conjuncts are denoted by the built-in constraint \( true \).

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

**Transition Relation**

\[ H'_1 \land H'_2 \land D \rightarrow (H_1 \land H_2 = H'_1 \land H'_2) \land G \land B \land H'_1 \land D \]

if \( (H_1 \setminus H_2 \leftrightarrow G \mid B) \) in \( P \) and \( CT \models D \rightarrow \exists x((H_1 \land H_2 = H'_1 \land H'_2) \land G) \)

A rule is applicable to CHR constraints, \( H' \), whenever these match (are instance of) the head atoms \( H \) of the rule, such that the guard \( G \) is satisfied, given the built-ins in the store. The matching is the effect of the existential quantification in \( \exists x(H = H') \), where \( x \) denotes the variables in the rule chosen from \( P \). Rule application is non-deterministic and committed-choice.

**Example.** The program below implements merge-sort\(^1\). The query \( \text{mergesort}(L) \) with \( L \) a list of length \( 2^n \), yields a tree-representation of the order.

\[
\begin{align*}
\text{true} \setminus \text{mergesort}([[]]) & \leftrightarrow \text{true}. \\
\text{true} \setminus \text{mergesort}([L|\text{List}]) & \leftrightarrow \text{root}(0, L), \text{mergesort}([\text{List}]). \\
\text{true} \setminus \text{root}(D, L_1), \text{root}(D, L_2) & \leftrightarrow \text{less}(L_1, L_2) \mid \text{root}(s(D), L_1), \text{edge}(L_1, L_2).
\end{align*}
\]

The first two rules decompose a list of elements, while adding new root/2 constraints to the store. The constraint \( \text{root}(D, L) \) 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 of the smallest.

3 A Transformational Approach to Termination Analysis

This section presents a termination preserving CHR to Prolog transformation.

\(^1\) Based on an example from http://www.cs.kuleuven.ac.be/~dtai/projects/CHR/
Transformation. The CHR constraint store is represented in Prolog as a list of constraints. This list is permuted before inspection to behave as a multiset. Each CHR rule is transformed into a rule/2 Prolog clause. The rule/2 predicate defines the transition relation between constraint lists.

A call to rule/2 with a list representation of the store, unifies the permutation of the list with a list representation of the rule’s head and a remainder $R$. If a unification exists such that all body atoms of the clause succeed, then the second term of rule/2 binds to the resulting store, by adding to $R$ the left heads and the introduced CHR constraints. I.e. a CHR rule of the form:

$$ H_1, ..., H_i \not \equiv H_j, ..., H_n \Leftrightarrow G_1, ..., G_k \mid B_{BI_1}, ..., B_{BI_i}, B_{CHR_1}, ..., B_{CHR_m}. $$

becomes a Prolog clause:

$$ \text{rule}([H_1, ..., H_i], [H_j, ..., H_n], B_{CHR_1}, ..., B_{CHR_m}, [R]) :- G_1, ..., G_k, B_{BI_1}, ..., B_{BI_i}. $$

Repeated rule application is captured by:\n
$$ \text{goal}(S) :- \text{permutation}(S, PS), \text{rule}(PS, NS), \text{goal}(NS). $$

Termination proofs of the transformed program take a general form, showing a decrease in level mapping: $[\text{goal}(S)] > [\text{goal}(NS)]$. This condition boils down to a condition on the norms of the arguments of all rule/2 clauses:

$$ ||PS||_{sl} > ||NS||_{sl}. $$

Here $|| \cdot ||_{sl}$ denotes the norm\(^2\) used on storelists.

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

$$ \text{rule}([\text{mergesort}([]), [R]], R), $$

$$ \text{rule}([\text{mergesort}(\text{List})], [R], \text{root}(\text{L}, \text{mergesort}(\text{List})) | R), $$

$$ \text{rule}([\text{root}(D, L_1), \text{root}(D, L_2) | R], \text{root}(\text{s}(D), L_1), \text{edge}(L_1, L_2), [R]) :- \text{less}(L_1, L_2). $$

The following conditions imply termination of mergesort:

$$ ||[\text{mergesort}([])], R ||_{sl} > ||R||_{sl}, $$

$$ ||[\text{mergesort}([\text{List}]), R]||_{sl} > ||[\text{root}(\text{L}, \text{mergesort}(\text{List})), R]||_{sl}, $$

$$ ||[\text{root}(D, L_1), \text{root}(D, L_2)|R] ||_{sl} > ||[\text{root}(\text{s}(D), L_1), \text{edge}(L_1, L_2)|R] ||_{sl}. $$

These conditions are met by the following storelist norm, $|| \cdot ||_{sl}$:

$$ ||[E_1, ..., E_n]||_{sl} = \sum_{i=1}^{n} ||E_i||_{c}. $$

Here, $|| \cdot ||_c$ are constraint norms:

$$ ||\text{mergesort}(\text{List})||_c = 1 + 2 \cdot ||\text{List}||_c, $$

$$ ||\text{root}(D, L)||_c = 1, $$

$$ ||\text{edge}(A, B)||_c = 0. $$

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

\(^2\) We have factored out permutation/2 to goal/1, as it is present in every rule clause.

\(^3\) The norm should be invariant over permutation.
4 Evaluation

We have implemented the transformation\(^4\) in K.U.Leuven CHR for SWI-Prolog. The implementation was used to transform two sets of CHR programs, one based on LP\(^5\) benchmarks and another from [2] marked with a ‘∗’. Polytool (PT) and AProVE (AP), based on respectively LP and TRS termination proof techniques, were used to prove termination of the transformed CHR programs:

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
& PT & AP & PT & AP & PT & AP \\
\hline
ackermann & - & + & convert & - & + & limpiece & - & + \\
average & + & + & diff & + & + & max & + & + & power & + & + \\
booland\(^*\) & + & + & factorial & + & + & mean & + & + & revlist & + & + \\
boolcard\(^*\) & - & + & gcd & + & + & mergesort & + & + & som & + & + \\
concat & - & - & joinlists & + & + & oddeven & + & + & weight & - & - \\
\hline
\end{array}
\]

Similar, but weaker results were obtained with the LP termination tools cTl and TerminWeb. 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.

5 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 [2] and can be automated 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 is not supported by our current transformation.

References

