A new approach to termination analysis of Constraint Handling Rules

Dean Voets, Paolo Pilozzi*, and Danny De Schreye

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

1 Introduction

Constraint Handling Rules (CHR) is a concurrent, committed-choice constraint programming language (see [2]). It is a rule-based language, in which multisets of atomic constraints are rewritten using guarded rules. It has a simple syntax and declarative semantics, and is very suitable for implementing constraint solvers.

Although the language is strongly related to Logic Programming (LP) and to lesser extent also to Term-Rewrite Systems (TRS), termination analysis of CHR has received little attention. To the best of our knowledge, the only contribution so far is reported in [3]. This study is limited to CHR programs with only one type of rules: the so called 'simplification rules'. The work shows that, for this class of programs, termination analysis techniques developed for LP (see [1]) can be adapted to CHR.

In this paper, we present a new approach to termination analysis of CHR which is applicable to a much larger class of CHR programs. We propose a new termination condition and show its applicability to CHR programs with rules that are not only of the simplification type. We have successfully tested the condition on a benchmark of programs, using a prototype analyser.

2 CHR: syntax and semantics

A constraint is a predicate in first-order logic. We distinguish between built-in (predefined) constraints and CHR (user-defined) constraints. Built-in constraints are handled by an underlying constraint solver. CHR constraints are defined by a CHR program. A CHR program is a finite set of CHR rules. There are three kinds of rules. *Simplification* rules replace a conjunction of constraints by an equivalent conjunction of (simplified) constraints. *Propagation* rules only generate new (redundant) constraints. These rules take respectively the form

$$H_1, ..., H_n \left(\begin{array}{c} \Leftrightarrow \\ \Rightarrow \end{array} \right) G_1, ..., G_k \mid B_1, ..., B_m.$$

where $H_1, ..., H_n$ is a conjunction of CHR constraints , $G_1, ..., G_k$ is a conjunction of (built-in) guards and $B_1, ..., B_m$ is a new conjunction of CHR constraints and built-in constraints. The language also offers *simpagation* rules, which represent a combination of simplification and propagation. We will not discuss this type of rule here any further, since it can be considered a redundant extension, aiming at more compact specifications.

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

Example 1 (Fibonacci).

$$fib(N, M1), fib(N, M2) \Leftrightarrow M1 = M2 \mid fib(N, M1).$$
$$fib(0, M) \Rightarrow M = 1.$$
$$fib(1, M) \Rightarrow M = 1.$$
$$fib(N, M) \Rightarrow N > 1 \mid N1 \; is \; N - 1, N2 \; is \; N - 2,$$
$$fib(N1, M1), fib(N2, M2), M \; is \; M1 + M2.$$

The operational semantics of CHR programs is given by a state transition system, where a state refers to a conjunction of (CHR and built-in) constraints. As mentioned above, we refer to it as the *constraint store*. A *computation* is a sequence of state transitions. A *query* provides the initial state. The computation terminates when an inconsistent state is obtained (failing computation) or when no more computation steps are possible. The selection of rules from the program is done in a fair way. However, it is a committed choice once the guard is satisfied.

Denoting by H, G and B; the conjunctions $H_1, ..., H_n$,$G_1, ..., G_k$ and $B_1, ..., B_m$ respectively, the state transition system $\rightarrow$ is specified as:

Simplify: $H' \wedge D \rightarrow (H' = H) \wedge G \wedge B \wedge D$
Propagate: $H' \wedge D \rightarrow (H' = H) \wedge H' \wedge G \wedge B \wedge D$

where H' is a conjunction of constraints in the constraint store which matches with the head H of some rule in the program, such that this matching, together with the guard G and the built-ins in B, are satisfiable in the underlying constraint solver, given the constraints in the remainder of the store, D. Note that all propagation rules can be activated infinitely often on the same conjunction of constraints, since the constraints that match with the head of the rule are not removed from the store. To avoid this, the operational semantics additionally includes a *propagation history*, which keeps track of which rules were applied on which conjunctions of constraints.

Example 2 (Fibonacci continued). With a typical query like $fib(3, N)$, the last propagation rule adds two new $fib/2$ constraints with lower first arguments to the store (but does not remove the constraint $fib(3, N)$), the first two propagation rules resolve base cases , while the simplification rule removes duplicates. The end state is the constraint store $fib(3, 3) \wedge fib(2, 2) \wedge fib(1, 1) \wedge fib(0, 1)$.

3 Termination analysis and CHR

In [3], termination proofs for CHR make use of a *ranking* function that maps constraints to natural numbers. It is the direct counterpart of the *norm* and *level mapping* used in LP (see [1]). One difference, however, is that in LP it suffices to reason about the rank or level values of individual atoms. In CHR, multiple constraints are simultaneously replaced by a number of constraints, or - for propagation rules - give rise to the addition of constraints. Although this implies different termination conditions, [3] shows that concepts and ideas from LP termination analysis are adaptable to CHR *with simplification only*.

The extension to programs with propagation rules gives a totally new termination problem. In LP, TRS and in CHR with simplification rules only, a

termination proof is based on a decreasing order associated to consecutive computation states. However, with each activation of a propagation rule in CHR, there is an explicit increase of the constraint store: new constraints are added and no existing constraints are removed. One would need to keep track of information regarding the propagation history *within* the computation states, in order to observe a decrease between consecutive states. Unfortunately, we expect that making the propagation history explicit in states could lead to rather messy and low-level termination conditions.

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 in the complete computation process, only a finite number of constraints are added to the store. Due to the use of the propagation history in CHR, this implies termination.

4 The ranking condition

In the following, P denotes a CHR program and I a query. constraints).

Definition 1 (Norm, level mapping). *A norm* $\| \, . \, \|$ *is a function from terms to* $\mathbb{N}$. *A level mapping* $| \, . \, |$ *is a function from CHR-constraints to* $\mathbb{N}$.

Definition 2 (Rigidity). *Let* C *be CHR constraint and* $| \, . \, |$ *a level mapping.* C *is rigid with respect to* $| \, . \, |$ *if for any substitution* $\theta : | \, C\theta \, | = | \, C \, |$.

Definition 3 (Call set). *Given a program* P *and a query* I*, the call set for* P *and* I*,* $Call(P, I)$*, is the set of all CHR constraints which enter the constraint store during a computation of* P *for* I.

Typically, in an automated analysis, we will use abstract interpretation to compute a safe (upper) approximation of the call set.

Definition 4 (Ranking condition for CHR). *A program* P *and a query* I *satisfy the Ranking condition for CHR, w.r.t. a level mapping* $| \, . \, |$ *if:*
1) All elements of $Call(P, I)$ *are rigid with respect to* $| \, . \, |$,
2) For each rule of P *and for each conjunction of atoms* H' *in* $Call(P, I)$ *which matches with the head* H*, using matching substitution* θ*, and such that all built-in constraints in* $G\theta$ *and* $B\theta$ *can be satisfied with answer substitution* θ'*:*

- *For a simplification rule* $H_1, ..., H_n \Leftrightarrow G \mid B_1, ..., B_m$*., with body-CHR constraints* $B_k, ..., B_m$*:*
  Let $m_h = max_{i=1,...,n} | \, H_i\theta \, |$ *and* $m_b = max_{j=k,...,m} | \, B_j\theta \, |$ *then*
  - *either* $m_h > m_b$ *, or,*
  - $m_h = m_b$ *and, with* q_h, q_b *the number of constraints with rank* m_h *in* $H_1\theta, ..., H_n\theta$ *and* $B_1\theta, ..., B_m\theta$ *respectively:* $q_h > q_b$
- *For a propagation rule* $H_1, ..., H_n \Rightarrow G \mid B_1, ..., B_m$*. , with body-CHR constraints* $B_k, ..., B_m$*:*
  - *for all* $i = 1, ..., n$ *and* $j = k, ..., m$*:* $| \, H_i\theta \, | > | \, B_j\theta \, |$.

Theorem 1 (Sufficiency of the Ranking condition). *Let* P *be a CHR program and* I *a query. If there exists a level mapping* $| \, . \, |$*, such that* P *and* I *satisfy the Ranking condition w.r.t.* $| \, . \, |$*, then all computations for* I *in* P *terminate.*

The idea of the proof is that, with the Ranking condition, indeed only finitely many constraints can ever enter the store. With propagation rules, only smaller ranking constraints can enter the store. Due to the propagation history, this process dies out. With simplification rules, either bigger ranking constraints are replaced by smaller ranking ones, or the number of maximally ranked ones is reduced. These effects combined yield only finitely many constraints.

Example 3 (Fibonacci continued). Let $fib(n, M)$ be a query, with n a natural number and M a free variable. One can infer that $\{fib(n, m) \mid n, m \in \mathbb{N}\} \cup \{fib(n, M) \mid n \in \mathbb{N}$ *and* M *a free variable*$\}$ is te call set. As a norm, we map each natural number to itself. The level mapping is defined on the call set as $\mid fib(n, X) \mid = \parallel n \parallel = n$. Clearly the call set is rigid w.r.t. $\mid . \mid$. For the first rule we have: $\mid fib(n, M1) \mid = \mid fib(n, M2) \mid = \mid fib(n, M1) \mid = n$. So, $m_h = m_b$ and $q_h > q_b$. The conditions for rules 2 and 3 are trivially satisfied, since they have no CHR constraints in their bodies. Finally, for rule 4, $\mid fib(n, M) \mid = n > \mid fib(n1, M1) \mid = n - 1$ and $\mid fib(n, M) \mid = n > \mid fib(n2, M2) \mid = n - 2$. Thus, the program terminates for these queries.

5 Experimental evaluation

We have implemented a prototype system using the proposed Ranking condition and tested it on a benchmark of CHR programs. The benchmark consists of a set of CHR programs based on LP termination problems, a set of CHR programs taken from [3], marked with a $'*'$, and some others taken from WebCHR (http://chr.informatik.uni-ulm.de/∼webchr/). The symbol $'+'$ denotes that the system proved termination, $'-'$ that it failed to do so. All programs terminate.

CHR, simplification only								General CHR	
ackermann	-	convert	+	linpoleq*	-	pathcons*	-	arccons*	-
average	-	diff	+	max	+	power	+	bool	+
binlog	+	factorial	+	mean	+	revlist	+	fibonacci	+
booland*	+	gcd	+	mergesort	+	som	+	integrity	+
boolcard*	+	genint	+	modulo	+	toyama	-	primes1	+
concat	-	joinlists	+	oddeven	+	weight	-	primes2	+

The success rate of the system is quite good, both for simplification only and for the general case. Note that in theory, the technique in [3] is more powerful on the simplification only cases. In order to be able to deal with propagation, our conditions on the simplification rules are stronger than those in [3]. However, it should be noted that [3] is a not available as an implemented system and that an implementation might actually not be able to support all examples of the theory (for instance if static analysis is used to infer some required information).

References

1. D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *Journal of Logic Programming*, 19-20:199–260, 1994.
2. T. Frühwirth. Theory and practice of Constraint Handling Rules. *Journal of Logic Programming*, 37(1–3):95–138, October 1998.
3. 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.