Scaling termination proofs by a characterisation of cycles in CHR

Paolo Pilozzi
paolo.pilozzi@cs.kuleuven.be [*]

Danny De Schreye
danny.deschreye@cs.kuleuven.be

CHR [3], short for Constraint Handling Rules, is a rule-based programming language, with similarities to Term Rewrite Systems (TRS) and Logic Programming (LP). In termination analysis of rule-based languages, the concept of a cycle is often useful. Informally, a cycle is a finite sequence of rules such that these rules can potentially be repeatedly executed during a computation. Above, we write “potentially” because cycles usually involve approximations. In cycles, whether one rule can actually activate the next is only verified locally (e.g. through matching or unification). But whether all these conditions hold simultaneously — so that the full cycle can be repeatedly traversed — is unknown.

Using cycles in termination analysis can have several advantages. They are mostly due to the fact that an analysis based on cycles is more modular. One advantage is reduced complexity. Some termination analysis techniques do not scale well for large programs. By performing the proof for separate cycles, sizes remain reduced. Another potential advantage is precision. When proving termination of a program as a whole, one typically selects a specific well-founded order. In the context of cycles, one can use a different order for each cycle. As a result, techniques based on cycles can be very successful (e.g. [4]).

We have studied the introduction of cycles in CHR. However, we were confronted with the fact that, due to the multi-headed rules and a multiset semantics in CHR, it is hard to define a suitable notion of “cycle”. Consider a CHR program with one rule, $a,c \Leftrightarrow a$, which allows to rewrite a conjunction $a \wedge c$ to a. Although one could consider it to be a cycle, in an actual computation it cannot result in an infinite loop, since every application removes a $c/0$ constraint. Similarly, a CHR program with one rule, $a,c,c \Leftrightarrow a,a,c$ is terminating and the rule would better not be characterised as a cycle. Considering a program with two rules, $a,a \Leftrightarrow b,b$ and $b \Leftrightarrow a$, we should identify a cycle. However, this cycle should contain the first rule together with two copies of the second rule.

One of our initial attempts to define cycles was very intuitive and rather precise, but it turned out that only for specific uses of a CHR program a concept of minimality existed, resulting in general in an infinite number of different minimal cycles. This is of course unacceptable. In this paper, we provide a less precise but useful definition of a cycle yielding a finite number of minimal cycles, and discuss its successful use in CHR termination analysis.

1 CHR syntax and semantics

First, we introduce the syntax and semantics of CHR [3]. The atoms of a CHR program, called *constraints*, are first-order relations on terms, collected in a *constraint store*. There are two kinds of constraints: *built-in constraints*, pre-defined in the host-language (CT, for Constraint Theory), and user-defined *CHR constraints*.

We assume presence of only one kind of CHR rule, that is, a *simplification rule*:

$$R_i \ @ \ H_{(i,1)},\dots,H_{(i,n_i)} \Leftrightarrow G_{(i,1)},\dots,G_{(i,u_i)} \mid B_{(i,1)},\dots,B_{(i,v_i)},C_{(i,1)},\dots,C_{(i,m_i)}.$$

Here, $H_{(i,1)},\dots,H_{(i,n_i)}$ are head constraints, representing the CHR constraints from the store that can be used to fire the rule. For rule application, given *matching* constraints for the heads, the built-in constraints $G_{(i,1)},\dots,G_{(i,u_i)}$ need to be satisfiable. Upon rule application, new built-in constraints, $B_{(i,1)},\dots,B_{(i,v_i)}$, and CHR constraints $C_{(i,1)},\dots,C_{(i,m_i)}$ are added to the constraint store, replacing the head constraints.

[*]Supported by I.W.T. Flanders - Belgium

Example 1 (Greatest Common Divisor). *The following CHR program, P, implements the Euclidean algorithm for computing the greatest common divisor (GCD).*

$$R_1 \ @ \ gcd(0) \Leftrightarrow true.$$
$$R_2 \ @ \ gcd(M), gcd(N) \Leftrightarrow N \geq M, M > 0 \mid L \ is \ N - M, gcd(M), gcd(L).$$

The first rule removes $gcd(0)$ constraints, while the second replaces two $gcd/1$ constraints by simpler $gcd/1$ constraints. □

The constraint store of a CHR program represents a conjunction of constraints in which multiple occurrences of the same constraint are allowed. The constraint store thus behaves as a *multiset*.

Definition 1 (Multiset). *A* multiset *is a tuple, $M_S = \langle S, m_S \rangle$, where S is a regular set, called the* underlying set *of elements, and m_S a* multiplicity function*, mapping the elements, e, of S to strict positive integer numbers, $m_S(e) \in \mathbb{N}_0$, representing the number of occurrences of e in M_S.* □

We may define m_S as its graph, i.e. the set of ordered pairs $\{(s, m_S(s)) : s \in S\}$. Therefore, the multiset written as $[\![a,a,b]\!]$ is defined as $\langle \{a,b\}, \{(a,2),(b,1)\} \rangle$, and the multiset $[\![a,b]\!]$ as $\langle \{a,b\}, \{(a,1),(b,1)\} \rangle$. For adding multiset together, we introduce *multiset join* and denote it by $\uplus$.

Since we do not consider propagation, the state transition system can be given by the *abstract CHR semantics*, where a *CHR state S* is a *constraint store* and only two kinds of transitions are defined:

1. For any state $[\![b]\!] \uplus S$, where b is a built-in constraint satisfiable in the host-language with substitution θ, a transition $([\![b]\!] \uplus S, S\theta)$ exists.

2. For any state $[\![h_1, \ldots, h_n]\!] \uplus S$, where $h_1, \ldots, h_n$ are CHR constraints matching with the head of a rule $H_1, \ldots, H_n \Leftrightarrow G_1, \ldots, G_k \mid B_1, \ldots, B_l, C_1, \ldots, C_m$ resulting in a substitution σ, such that the built-in constraints $G_1\sigma, \ldots, G_k\sigma$ are satisfiable in the host-language (CT) with substitution θ, a transition $([\![h_1, \ldots, h_n]\!] \uplus S, ([\![B_1, \ldots, B_l, C_1, \ldots, C_m]\!] \uplus S)\sigma\theta)$ exists.

We assume that any call to the host-language can only introduce variable bindings and must do so in finite time. If a built-in constraint cannot be solved, the CHR program fails. CHR is furthermore a committed choice language. Therefore, whenever a rule is applied — which is a non-deterministic choice between all possible applications of the CHR rules of the program — we commit on this choice.

2 CHR cycles

The rules of a CHR program relate CHR constraints. The head constraints of a rule represent the constraints *required* for rule application. The added CHR constraints represent the constraints *available* after rule application. To denote these sets of constraints, we introduce *abstract CHR constraints*.

Definition 2 (Abstract CHR constraint). *An abstract CHR constraint $\mathscr{C} = (C, B)$ is a pair, where C is a CHR constraint and B a conjunction of built-in constraints. Given a mapping $\wp : (C,B) \mapsto \{C\sigma \mid \exists\theta : CT \models B\sigma\theta\}$, an abstract CHR constraint represents a set of CHR constraints.* □

Two kinds of abstract CHR constraints are present in a CHR program. An *abstract input constraint $in_{(i,j)} = (H_{(i,j)}, G_i)$*, where $G_i = G_{(i,1)} \wedge \ldots \wedge G_{(i,u_i)}$, represents a set of CHR constraints matching with the head $H_{(i,j)}$, possibly resulting in rule application. An *abstract output constraint $out_{(i,j)} = (C_{(i,j)}, G_i)$* represents the CHR constraints which become available after rule application.

Example 2 (GCD cont.). *In the following, we denote by $\mathscr{I}n_{R_i}$ and $\mathscr{O}ut_{R_i}$ the multisets of abstract input constraints and output constraints of a rule R_i, and by $\mathscr{I}n_P$ and $\mathscr{O}ut_P$, the multisets of abstract input and output constraints of a set of rules, P. For the GCD program in Example 1, we obtain:*

$$\begin{array}{lll} \mathscr{I}n_P & = & \mathscr{I}n_{R_1} \uplus \mathscr{I}n_{R_2} = [\![in_{(1,1)}, in_{(2,1)}, in_{(2,2)}]\!], \textit{ where} \\ & & in_{(1,1)} = (gcd(0), true) \\ & & in_{(2,1)} = (gcd(M), N \geq M \wedge M > 0) \\ & & in_{(2,2)} = (gcd(N), N \geq M \wedge M > 0) \\ \mathscr{O}ut_P & = & \mathscr{O}ut_{R_1} \uplus \mathscr{O}ut_{R_2} = [\![out_{(2,1)}, out_{(2,2)}]\!], \textit{ where} \\ & & out_{(2,1)} = (gcd(M), N \geq M \wedge M > 0) \\ & & out_{(2,2)} = (gcd(L), N \geq M \wedge M > 0) \end{array} \qquad \square$$

The rules of a CHR program relate abstract inputs to abstract outputs.

Definition 3 (Rule transition relation)**.** *A rule transition of a CHR program P is an ordered pair $T_i = (\mathscr{I}n_{R_i}, \mathscr{O}ut_{R_i})$, relating the multiset of abstract input constraints $[\![in_{(i,1)}, \ldots, in_{(i,n_i)}]\!] = \mathscr{I}n_{R_i}$ of a rule $R_i \in P$ to the multiset of abstract output constraints $[\![out_{(i,1)}, \ldots, out_{(i,m_i)}]\!] = \mathscr{O}ut_{R_i}$ of R_i. The rule transition relation $\mathscr{T} = \{T_i \mid R_i \in P\}$ of a CHR program P is the set of rule transitions T_i of a program P.* $\square$

Example 3 (GCD cont.)**.** *The GCD program has two rules and thus two rule transitions:*

$$\mathscr{T} = \{T_1, T_2\}, \textit{ where } T_1 = ([\![in_{(1,1)}]\!], [\![]\!]) \textit{ and } T_2 = ([\![in_{(2,1)}, in_{(2,2)}]\!], [\![out_{(2,1)}, out_{(2,2)}]\!]). \qquad \square$$

Abstract outputs relate to inputs by a *match transition relation*, given by a dependency analysis.

Definition 4 (Match transition relation)**.** *A match transition of a CHR program P is an ordered pair $M_{(i,j,k,l)} = (out_{(i,j)}, in_{(k,l)})$, expressing a dependency between an output $out_{(i,j)} = (C_{(i,j)}, G_i)$ of $\mathscr{O}ut_P$ and an input $in_{(k,l)} = (C_{(k,l)}, G_k)$ of $\mathscr{I}n_P$, for which $\exists\theta : CT \models ((C_{(i,j)} = C_{(k,l)}) \wedge G_i \wedge G_k)\theta$ holds. The match transition relation $\mathscr{M}$ is the set of all match transitions $M_{(i,j,k,l)}$ in P.* $\square$

Notice that a match transition exists if the output and input it connects represent sets of constraints with a non-empty intersection: $\wp(out_{(i,j)}) \cap \wp(in_{(k,l)}) \neq \emptyset$.

Example 4 (GCD cont.)**.** *We obtain for the GCD program the following match transition relation:*

$$\mathscr{M} = \{M_{(2,1,2,1)}, M_{(2,1,2,2)}, M_{(2,2,1,1)}, M_{(2,2,2,1)}, M_{(2,2,2,2)}\}.$$

Note that the first output of the second rule does not match with the input of the first rule as their intersection $\wp(out_{(2,1)}) \cap \wp(in_{(1,1)}) = \emptyset$. $\square$

For a program P, we call $\mathscr{N}_P = \langle \mathscr{I}n_P, \mathscr{O}ut_P, \mathscr{T}_P, \mathscr{M}_P \rangle$ its *CHR net*.

2.1 CHR cycles

To discuss cycles, we need an abstraction for computations, one for which a concept of minimality exists. For this purpose, we introduce *CHR constructs*. A CHR construct is an ordered pair of multisets. The first multiset represents the number of applications of some rule in a subcomputation of a program. The second multiset represents matches within the subcomputation. Note that if a *universe* U, in which the elements of the multiset $M_S = \langle S, m_S \rangle$ must live, is specified, that we can replace m_S by a function $\mu_S : U \rightarrow \mathbb{N}$, obtained by extending m_S to U with values 0 outside S.

Definition 5 (Cyclic CHR construct)**.** *Let $\mathscr{N}_P = \langle \mathscr{I}n_P, \mathscr{O}ut_P, \mathscr{T}_P, \mathscr{M}_P \rangle$ be the CHR net of P. Then, a* CHR construct *is a pair of multisets $((P, \mu), (\mathscr{M}_P, \mu'))$: a multiset of rules (P, μ) and a multiset of match transitions $(\mathscr{M}_P, \mu')$; such that:*

$$\forall in_{(i,j)} \in \mathscr{I}n_P : \sum_{M_{(k,l,i,j)} \in \mathscr{M}_P} \mu'(M_{(k,l,i,j)}) \leq \mu(R_i) \qquad \forall out_{(i,j)} \in \mathscr{O}ut_S : \sum_{M_{(i,j,k,l)} \in \mathscr{M}_P} \mu'(M_{(i,j,k,l)}) \leq \mu(R_i)$$

A cyclic CHR construct *or* CHR cycle *is a CHR construct where additionally*

$$\forall in_{(i,j)} \in \mathscr{I}n_C : \sum_{M_{(k,l,i,j)} \in \mathscr{M}_P} \mu'(M_{(k,l,i,j)}) = \mu(R_i).$$

We say that a CHR construct is traversed, if every rule in it was applied once according to the match transitions of the construct. □

Notice that for a CHR construct to correspond to an actual computation, we need to constrain the number of outgoing match transitions for a given output to the number of times its rule occurs in the multiset of rule applications. Similarly, we need to constrain the number of incoming match transitions for a given input to the number of times its rule occurs in the multiset of rule applications. This becomes clear when regarding the next example computation for the GCD program from Example 2.

Example 5 (GCD cont.)**.** *Given a query $I = [gcd(6), gcd(3)]$, the following sequence represents a computation of P for I.*

$$[gcd(6), gcd(3)] \mapsto_{R_2} [gcd(3), gcd(3)] \mapsto_{R_2} [gcd(0), gcd(3)] \mapsto_{R_1} [gcd(3)]$$

A different representation of this computation uses concepts of CHR nets, as is shown in Figure 1. The

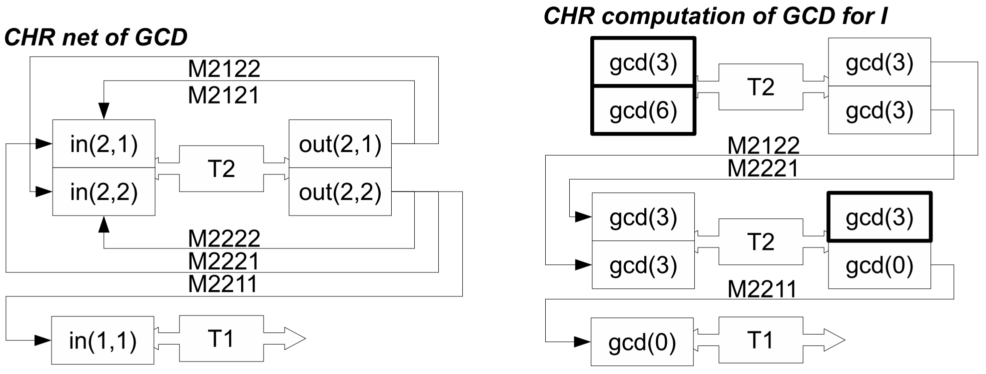

Figure 1: CHR net of the GCD program and a possible computation of GCD for *I*.

inputs without an incoming match transition, in the CHR construct representing a computation of P for I, represent the constraints from the query . Outputs without an outgoing match transition represent the constraints part of the answer. □

Finally, a CHR construct is cyclic if all inputs are provided for from within the construct. Thus, a cyclic construct replaces the constraints which are removed when traversing the construct.

The set of all possible (cyclic) CHR constructs can be represented using a system of linear inequalities. Such a system has a unique computable basis of solutions, called a *Hilbert basis*. From it, we can compute any solution to the system by positive linear combinations of the solution vectors.

Example 6 (GCD cont.)**.** *For the GCD program, we obtain the following system of inequalities, describing the cyclic CHR constructs of the program. Note that m_{R_i} represents $\mu(R_i)$, the number of rule applications of R_i in the construct, and that $m_{(i,j,k,l)}$ represents $\mu'(M_{(i,j,k,l)})$ the number of match transitions $M_{(i,j,k,l)}$ in the construct.*

$$\begin{aligned} m_{(2,2,1,1)} &= m_{R_1} & \qquad m_{(2,1,2,1)} + m_{(2,1,2,2)} &\leq m_{R_2} \\ m_{(2,1,2,1)} + m_{(2,2,2,1)} &= m_{R_2} & m_{(2,2,2,1)} + m_{(2,2,2,2)} &\leq m_{R_2} \\ m_{(2,1,2,2)} + m_{(2,2,2,2)} &= m_{R_2} & & \end{aligned}$$

By adding slack variables we obtain a system of equations with positive integer solutions, which allows us to compute the following Hilbert basis.

$$([\![R_2]\!],[\![M_{(2,1,2,2)},M_{(2,2,2,1)}]\!]) \qquad ([\![R_2]\!],[\![M_{(2,1,2,1)},M_{(2,2,2,2)}]\!])$$

We can obtain every cyclic construct of a CHR program by joining elements from the basis (Figure 2). □

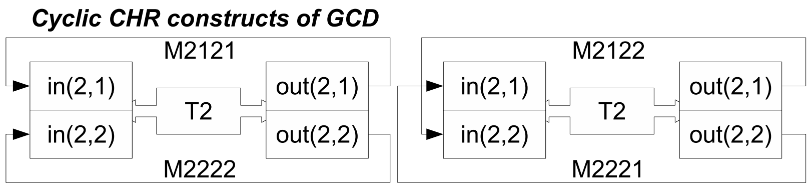

Figure 2: Cyclic CHR constructs of the GCD program.

By considering now the cycles in the basis (See Figure 2), we can derive the subsets of rules that demonstrate cyclic behaviour. For the GCD example, which is terminating for all queries, there is only one such subset, $\{R_2\}$, that is cyclic. So it suffices to consider only this cycle to investigate termination.

3 Evaluation and conclusion

We described an approach for analysing the cycles of a CHR program, as such safely over-approximating the loops of the program. For termination, it is sufficient to analyse only cycles and not the entire program, resulting in faster and more accurate termination proofs. We have developed an implementation for deriving the CHR net of a program, for performing a Strongly Connected Component (SCC) analysis on the CHR net of the program, and for generating systems of linear equations for each of the SCCs.

Our approach for cycle analysis in CHR is useful, however, only when regarding sufficiently large programs, consisting of multiple cyclic subsets of rules. Our approach scales well, using first a SCC analysis to make an initial estimate of the possible cycles of a program. Then, only afterwards for the remaining components, a classification is obtained by computing the basis [5] of solutions. If empty, the component is not a cycle, otherwise, it is a cycle and is analysed for termination.

Our representation of cycles can be used for more refined approaches to termination analysis of CHR as well, e.g. approaches concerning dependency pairs [1] and Ramsey's theorem [2]. Future work will involve investigating whether these approaches to termination analysis can be ported to the CHR context.

References

[1] Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Electronic Notes in Theoretical Computer Science*, 236(1-2):133–178, 2000.

[2] Nachum Dershowitz, Naomi Lindenstrauss, Yehoshua Sagiv, and Alexander Serebrenik. A General Framework for Automatic Termination Analysis of Logic Programs. *CoRR*, cs.PL/0012008, 2000.

[3] Thom Frühwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.

[4] Manh Thang Nguyen, Jürgen Giesl, Peter Schneider-Kamp, and Danny De Schreye. Termination analysis of logic programs based on dependency graphs. pages 8–22, 2007.

[5] Dmitrii V. Pasechnik. On computing the Hilbert base via the Elliott-MacMahon algorithm. *Electronic Notes in Theoretical Computer Science*, 263(1-2):37–46, 2001.