Turing-complete Subclasses of CHR

Jon Sneyers

K.U. Leuven, Belgium

`jon.sneyers@cs.kuleuven.be`

Abstract. Interesting subclasses of CHR are still Turing-complete: CHR with only one kind of rule, with only one rule, and propositional refined CHR. This is shown by programming a simulator of Turing machines (or Minsky machines) within those subclasses. Single-headed CHR without host language and propositional abstract CHR are not Turing-complete.

1 Introduction

As a stand-alone programming language, CHR is Turing-complete [8]. In this paper we demonstrate that several subclasses of CHR are also Turing-complete. Section 2 shows that CHR with only one kind of rules (simplification, propagation, or simpagation) is still Turing-complete. Then, in Section 3, we see that multiple rules are not needed — one single rule suffices. Finally, Section 4 deals with single-headed CHR and propositional CHR (only zero-arity constraints).

Some very restricted subclasses of CHR turn out to be not Turing-complete. Propositional CHR is not Turing-complete under the abstract semantics, so allowing constraints to have (even only non-compound) arguments does add computational power. Single-headed CHR without host language is not Turing-complete either, so computational power is gained by the availability of either multi-headed rules or a host language. Finally, the added execution control of a more instantiated operational semantics (e.g. the refined semantics [4] or the priority semantics [2]) also adds computational power in the sense that propositional CHR becomes Turing-complete under a more instantiated semantics.

Because of space constraints, we need to assume that the reader is familiar with CHR [5], Turing machines [9], and Minsky machines [7]. Unless otherwise mentioned, we assume that the abstract operational semantics ω_t is used.

2 Only One Kind of Rules

The TMSIM program of Listing 1 (adapted from [8]) contains only simpagation rules. It can easily be modified to consist only of simplification rules: the kept part of the simpagation rules can be removed and reinserted in the body.

It is also possible to simulate Turing machines in CHR with only propagation rules. Since there is no way to destructively update the constraints, we add a time-stamp argument to all constraints that are updated. By carefully making sure that all the non-modified tape cells and adjacency constraints are copied

Listing 1. TMSIM: Turing machine simulator

```
r1 @ delta(Q,S,P,T,left), adj(L,C) \ state(Q), cell(C,S), head(C)
    <=> state(P), cell(C,T), head(L).
r2 @ delta(Q,S,P,T,right), adj(C,R) \ state(Q), cell(C,S), head(C)
    <=> state(P), cell(C,T), head(R).
r3 @ delta(Q,S,P,T,left) \ left(C), state(Q), cell(C,S), head(C)
    <=> cell(L,b), left(L), adj(L,C), state(P), cell(C,T), head(L).
r4 @ delta(Q,S,P,T,right) \ right(C), state(Q), cell(C,S), head(C)
    <=> cell(R,b), adj(C,R), right(R), state(P), cell(C,T), head(R).
```

Listing 2. TMSIM-2R: Turing machine simulator in two rules

```
% left(C) <=> adj(L,L), adj(L,C), cell(L,b).
% right(C) <=> adj(R,R), adj(C,R), cell(R,b).

r13 @ delta(Q,S,P,T,left), state(Q), head(C) \ adj(L,C), adj(C,R), cell(C,S)
    <=> adj(L,C2), adj(C2,R), adj(C,C), cell(C,b), cell(C2,T), state(P), head(L).
r24 @ delta(Q,S,P,T,right), state(Q), head(C) \ adj(L,C), adj(C,R), cell(C,S)
    <=> adj(L,C2), adj(C2,R), adj(C,C), cell(C,b), cell(C2,T), state(P), head(R).
```

from one time-stamp to the next, and the modified cell (and modified adjacency constraints in case of a tape extension) is not copied, the correct behavior can be obtained without destructive updates. Since the entire tape is copied at every step, the space used by such a propagation-rule-only program is $O(TS)$ when simulating a T -time, S -space Turing machine. This is considerably less efficient than the $O(S)$ space usage of the TMSIM program.

3 Only One Rule

Now let us see how many rules are really needed. In Listing 1, we have two rules for each direction of tape movement (**left** and **right**): one for the normal case and one for the case in which the tape representation needs to be extended. Using a slightly different tape representation, these two cases can be merged.

The idea is as follows. The tape is still represented using adjacency constraints **adj/2**, but instead of marking the tape ends with **left/1** and **right/1**, we put an extra blank cell with a loop to itself at both ends, as shown in Fig. 1.

When moving to the right (the case for moving to the left is symmetric), we create a new cell and add two **adj/2** constraints as shown in Fig. 2(a). We overwrite the current cell symbol with the blank symbol **b** and remove the incoming and outgoing **adj/2** constraints. Finally we add a looping **adj/2** constraint at the current cell. In the normal case (Fig. 2(a)) the net result is that the current cell becomes isolated and a tape ‘bypass’ is created. When we are at the tape end, the net result is that the tape has been extended with one cell, and we remain at the (new) tape end (see Fig. 2(b)). In listing 2 the resulting CHR program TMSIM-2R is shown; it has only two rules. The two rules that are commented out show how to change the old tape representation to the new form, but we assume that the query is already in the correct form.

Fig. 1. Tape representation for TMSIM-2R

Fig. 2. Two possibilities for the right move in TMSIM-2R

Now can we further reduce this program? Certainly! It is easy to see that there is a lot of symmetry between the two rules in TMSIM-2R. We can change the tape representation again to take advantage of this symmetry and merge the rules. Instead of using $\text{adj}(A,B)$ to represent that cell A is to the left of cell B , we now use the following redundant representation: $\text{adj}(A,B,\text{left})$, $\text{adj}(B,A,\text{right})$. So $\text{adj}(A,B,X)$ means that cell A is to the X of cell B . By replacing every $\text{adj}/2$ constraint in the two-rule program TMSIM-2R by the corresponding two $\text{adj}/3$ constraints, and by connecting the third argument of the $\text{adj}/3$ with the last argument of the $\text{delta}/5$ constraint, both rules in TMSIM-2R become a variable renaming of each other, so only one of them is needed. Listing 3 shows the resulting CHR program TMSIM-1R.

4 Only Single-Headed Rules and Propositional CHR

The program TMSIM-1R consists of only one rule, but that rule has 8 head constraints. All rules with more than two heads can be reduced to several rules with only two heads by explicitly constructing partial joins. However, as we shall see shortly, single-headed rules only do not suffice.

All of the above programs are CHR programs without host language: no host language predicates are used in rule guards or bodies, and constraint arguments are only constants and variables. An arbitrary number of fresh variables can be created if the body of a rule contains variables that do not occur in the head, but all variables are syntactically different objects and they cannot be bound to each other or to constants (as in Prolog). CHR with only single-headed rules can be shown to be Turing-complete if some form of host language arithmetic is allowed [3]. Host language arithmetic can be done directly using integers, or successor term notation can be used, so an argument of the form

Listing 3. TMSIM-1R: Turing machine simulator in one rule

```
% adj(A,B) <=> adj(A,B,left), adj(B,A,right).

r1234 @ delta(Q,S,P,T,D), state(Q), head(C) \ adj(A,C,D), adj(C,B,D),
      cell(C,S), adj(C,A,E), adj(B,C,E)
<=> adj(A,C2,D), adj(C2,B,D), adj(C,C,D), cell(C,b), cell(C2,T),
      adj(C2,A,E), adj(B,C2,E), adj(C,C,E), state(P), head(A).
```

$s(s(s(\dots s(\text{zero})\dots)))$, with n times s , indicates the number n . In both cases however, a nontrivial host language data type is needed — respectively integers and complex terms. Alternatively, constraint chains can be used to represent numbers, but then either multi-headed rules are needed, or a form of assignment (for example Prolog unification).

Without a host language, CHR with only single headed rules is a very weak formalism — it is not more powerful than propositional CHR, i.e. CHR with only zero-arity constraints:

Lemma. *Single-headed CHR without host language can be reduced to propositional CHR without host language.*

Proof (sketch). Assume there are k distinct constants used in the program and queries. For every CHR constraint c/n , we make “flattened” versions of it, one for every possible combination of arguments. Since variables cannot be bound and since in single-headed rules there can only be matching between the arguments of a single constraint, it does not matter what the variables are, but only which ones are the same. Hence, there are less than $(k + n)^n$ different combinations of arguments for a constraint of arity n . We can replace the body CHR constraints by their flattened version, and duplicate the rule for every matching flattened version of the head. The resulting program is propositional. $\square$

We have the following negative result for propositional CHR-only:

Theorem. *Propositional CHR (without host language) is not Turing-complete.*

Proof (sketch). Propositional CHR programs can easily be encoded as place-transition Petri nets [1], which are not Turing-complete [6]. $\square$

The above lemma and theorem can be combined in the obvious way:

Corollary. *Single-headed CHR without host language is not Turing-complete.*

Somewhat surprisingly, the above theorem only holds when the full abstract semantics ω_t is considered. If the refined operational semantics ω_r [4] is used, the result no longer holds. The reason is that the ω_r semantics allows checking for the *absence* of a constraint, which is not possible in the ω_t semantics. Consider the following encoding of Minsky machines in propositional refined CHR:

$$\begin{array}{ll} \llbracket p_i : \text{succ}(r_1) \rrbracket = & p_i \text{ <=> } a, p_{i+1}. \\ \llbracket p_i : \text{succ}(r_2) \rrbracket = & p_i \text{ <=> } b, p_{i+1}. \end{array}$$

$$\begin{aligned}
\llbracket p_i : \text{decjump}(r_1, p_j) \rrbracket &= & p_i, \mathbf{a} <=> p_{i+1}. \\
&& p_i <=> p_j. \\
\llbracket p_i : \text{decjump}(r_2, p_j) \rrbracket &= & p_i, \mathbf{b} <=> p_{i+1}. \\
&& p_i <=> p_j.
\end{aligned}$$

It crucially depends on the refined semantics. The value of register r_1 is represented by the multiplicity of the constraint $\mathbf{a}$, and the value of r_2 by the multiplicity of $\mathbf{b}$. Under the abstract semantics this program breaks down because the second rule for **decjump** instructions can be fired even if the corresponding register does not have the value zero — this cannot happen in the refined semantics because of the rule order. Thus, propositional CHR is not Turing-complete in the abstract semantics, but it is Turing-complete in the refined semantics. In the priority semantics ω_p [2], propositional CHR is also Turing-complete: it suffices to assign a higher priority to the first rule for **decjump** than to the second rule. We conjecture that *single-headed* CHR without host language remains not Turing-complete even under the ω_r semantics.

5 Future Work

These surprising results indicate the need for further and deeper investigation of the connection between language features, operational semantics, termination and confluence, and computability and expressivity.

Acknowledgments. I thank Thom Frühwirth, Leslie De Koninck, and Bart Demoen for the discussions that lead to some of the ideas presented here. This research was funded by a Ph.D. grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen).

References

1. Hariolf Betz. Relating coloured Petri nets to Constraint Handling Rules. In *CHR '07*, pages 33–47, Porto, Portugal, 2007.
2. Leslie De Koninck, Tom Schrijvers, and Bart Demoen. User-definable rule priorities for CHR. In *PPDP '07*, pages 25–36, Wrocław, Poland, 2007. ACM Press.
3. Cinzia Di Giusto, Maurizio Gabbriellini, and Maria Chiara Meo. Expressiveness of multiple heads in CHR. *CoRR*, abs/0804.3351, 2008.
4. Gregory J. Duck, Peter J. Stuckey, María García de la Banda, and Christian Holzbaaur. The refined operational semantics of Constraint Handling Rules. In *ICLP '04*, volume 3132 of *LNCS*, pages 90–104, Saint-Malo, France, 2004. Springer.
5. Thom Frühwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.
6. Matthias Jantzen and Rüdiger Valk. Formal properties of place/transition nets. In *Advanced Course: Net Theory and Applications*, volume 84 of *LNCS*, Hamburg, Germany, 1980. Springer.
7. Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice Hall, 1967.
8. Jon Sneyers, Tom Schrijvers, and Bart Demoen. The computational power and complexity of Constraint Handling Rules. To appear in ACM TOPLAS, 2008.
9. Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Mathematical Society*, 2(42):230–265, 1936.