

Expressiveness of Multiple Heads in CHR

Cinzia Di Giusto Maurizio Gabbrielli Maria Chiara Meo

SOFSEM, Šplinderův Mlýn 2009

CHR

CHR

Constraint Handling Rules is a high-level programming language based on multi-headed, committed-choice, guarded multiset rewrite rules.

Thom Frühwirth

Guarded Multiset Rewriting rules

Rule

$$H \Leftrightarrow C \mid B$$

Informally: if the guard C is satisfied replace the constraints in H with the ones in B .

CHR by example

Program: Less than or equal \leq

reflexivity @ $LEQ(X, Y) \Leftrightarrow X = Y \mid \text{true}$

antisymmetry @ $LEQ(X, Y), LEQ(Y, X) \Leftrightarrow \text{true} \mid X = Y$

transitivity @ $LEQ(X, Y), LEQ(Y, Z) \Rightarrow \text{true} \mid LEQ(X, Z)$

Example

- Run the above program on $LEQ(A, B), LEQ(B, C), LEQ(C, A)$

CHR by example

Program: Less than or equal \leq

reflexivity @ $LEQ(X, Y) \Leftrightarrow X = Y \mid \text{true}$

antisymmetry @ $LEQ(X, Y), LEQ(Y, X) \Leftrightarrow \text{true} \mid X = Y$

transitivity @ $LEQ(X, Y), LEQ(Y, Z) \Rightarrow \text{true} \mid LEQ(X, Z)$

Example

- Run the above program on $LEQ(A, B), LEQ(B, C), LEQ(C, A)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C)$

CHR by example

Program: Less than or equal \leq

reflexivity @ $LEQ(X, Y) \Leftrightarrow X = Y \mid \text{true}$

antisymmetry @ $LEQ(X, Y), LEQ(Y, X) \Leftrightarrow \text{true} \mid X = Y$

transitivity @ $LEQ(X, Y), LEQ(Y, Z) \Rightarrow \text{true} \mid LEQ(X, Z)$

Example

- Run the above program on $LEQ(A, B), LEQ(B, C), LEQ(C, A)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, A)$

CHR by example

Program: Less than or equal \leq

reflexivity @ $LEQ(X, Y) \Leftrightarrow X = Y \mid \text{true}$

antisymmetry @ $LEQ(X, Y), LEQ(Y, X) \Leftrightarrow \text{true} \mid X = Y$

transitivity @ $LEQ(X, Y), LEQ(Y, Z) \Rightarrow \text{true} \mid LEQ(X, Z)$

Example

- Run the above program on $LEQ(A, B), LEQ(B, C), LEQ(C, A)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, A)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, A), LEQ(C, B)$

CHR by example

Program: Less than or equal \leq

reflexivity @ $LEQ(X, Y) \Leftrightarrow X = Y \mid \text{true}$

antisymmetry @ $LEQ(X, Y), LEQ(Y, X) \Leftrightarrow \text{true} \mid X = Y$

transitivity @ $LEQ(X, Y), LEQ(Y, Z) \Rightarrow \text{true} \mid LEQ(X, Z)$

Example

- Run the above program on $LEQ(A, B), LEQ(B, C), LEQ(C, A)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, A)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, A), LEQ(C, B)$
- Symmetry rule: $LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, C), A = B$

CHR by example

Program: Less than or equal \leq

reflexivity @ $LEQ(X, Y) \Leftrightarrow X = Y \mid \text{true}$

antisymmetry @ $LEQ(X, Y), LEQ(Y, X) \Leftrightarrow \text{true} \mid X = Y$

transitivity @ $LEQ(X, Y), LEQ(Y, Z) \Rightarrow \text{true} \mid LEQ(X, Z)$

Example

- Run the above program on $LEQ(A, B), LEQ(B, C), LEQ(C, A)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, A)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, A), LEQ(C, B)$
- Symmetry rule: $LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, C), A = B$
- Symmetry rule: $LEQ(B, C), LEQ(B, C), A = B, A = C$

CHR by example

Program: Less than or equal \leq

reflexivity @ $LEQ(X, Y) \Leftrightarrow X = Y \mid \text{true}$

antisymmetry @ $LEQ(X, Y), LEQ(Y, X) \Leftrightarrow \text{true} \mid X = Y$

transitivity @ $LEQ(X, Y), LEQ(Y, Z) \Rightarrow \text{true} \mid LEQ(X, Z)$

Example

- Run the above program on $LEQ(A, B), LEQ(B, C), LEQ(C, A)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, A)$
- Transitivity rule: $LEQ(A, B), LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, A), LEQ(C, B)$
- Symmetry rule: $LEQ(B, C), LEQ(C, A), LEQ(A, C), LEQ(B, C), A = B$
- Symmetry rule: $LEQ(B, C), LEQ(B, C), A = B, A = C$
- Symmetry rule: $A = B, A = C, B = C$

Our work

Claim

In [T. Frühwirth. Theory and practice of constraint handling rules, 1998] it was claimed that in the *LEQ* program the presence of multiple heads is **necessary**: i.e. a single headed multiset rewriting rules language **cannot** exhibit the same behaviour.

Our Contribution

We formally prove that multiple heads **augment** the expressive power of CHR.

Outline

- 1 The Language
- 2 On the Turing Completeness of CHR
- 3 Separation Results
- 4 Conclusions

The language: formal definition (1)

Definition (Constraints)

Two types of constraints:

- **User defined constraints**: constraints defined by the CHR program (*LEQ*)
- **Built-in constraints**: we assume a given first order theory which describes their meaning (=)

The language: formal definition (2)

Definition (Syntax)

A CHR **simplification** rule has the form: $r @ H \Leftrightarrow C \mid B$

A CHR **propagation** rule has the form: $r @ H \Rightarrow C \mid B,$

A CHR **program** is a set of CHR simplification and propagation rules.

A CHR **goal** is a multiset of user-defined and built-in constraints.

Operational Semantics

- We use a transition system $T = (Conf, \longrightarrow)$ where configurations are triples $\langle G, K, d \rangle$,
 - G are the constraints of the goal that remain to be solved,
 - K are the user-defined constraints that have been accumulated
 - d are the built-in constraints that have been simplified.
- An **initial configuration** has the form $\langle G, \emptyset, \emptyset \rangle$.
- A **final configuration** has either the form $\langle G, K, false \rangle$ when it is *failed*, or the form $\langle \emptyset, K, d \rangle$ when it is successfully terminated because there are no applicable rules.

Observables (1)

Definition (Data Sufficient Answer)

Let P be a CHR program and G a goal, a **Data Sufficient Answer** $\mathcal{SA}_P(G)$ is the multiset that can be observed for computations that terminate with an empty goal and an empty user-defined constraint $\langle\langle\emptyset, \emptyset, d\rangle\rangle$.

- The answer $A = B, B = C, C = A$ we have obtained for the program LEQ is a data sufficient answer

Observables (2)

Definition (Qualified Answers)

Let P be a CHR program and G a goal, a **Qualified Answer** $QA_P(G)$ is the multiset that can be observed for computations that terminate with an empty goal and a non necessary empty user-defined constraint $(\langle \emptyset, K, d \rangle)$.

- Clearly Qualified Answers are a superset of Data Sufficient answers.

CHR is Turing equivalent

Theorem

If the underlying CT contains at least one function symbol of arity one and one constant, CHR with single heads only is Turing powerful

Hints on the proof

- The presence of a function symbol and a constant permit to encode natural numbers
- Hence it is possible to provide an encoding of Minsky machines

But...

Theorem

If the underlying CT contains only a finite of constant symbols, CHR with single heads only is not Turing complete

Hints on the proof

- Indeed CHR-s with CT_{\emptyset} is computationally equivalent to Petri nets

A First Separation between CHR-s and CHR

Theorem

If the underlying CT contains only a finite of constant symbols, CHR (with multiple heads) is Turing complete

Hints on the proof

- It is possible to provide an encoding of Minsky machines
- the key idea is that natural numbers are encoded using chains of atomic formulas: $s(R_1, SuccR_1)$, $s(SuccR_1, SuccR'_1) \dots$ and R_1 , $SuccR_1$, $SuccR'_1 \dots$ are variables.

Remarks

- Previous theorems state a separation result between CHR and CHR-s
- Real implementations consider non-trivial CT
- We are interested in finer separation results

Preliminaries

- If CT is non-trivial CHR and CHR-s are Turing equivalent.
- In principle one can always encode CHR into CHR-s.
- But how difficult or how acceptable is such an encoding?

Acceptable encodings

Definition (Acceptable encoding)

An **acceptable encoding** (of CHR into CHR-s, for the class of goals \mathcal{G}) is a pair of mappings $\llbracket \cdot \rrbracket : \mathcal{P}_{CHR} \rightarrow \mathcal{P}_{CHR-s}$ and $\llbracket \cdot \rrbracket_g : \mathcal{G}_{CHR} \rightarrow \mathcal{G}_{CHR-s}$ which satisfy the following conditions:

- \mathcal{P}_{CHR} and \mathcal{P}_{CHR-s} share the same CT;
- for any goal $(A, B) \in \mathcal{G}_{CHR}$, $\llbracket A, B \rrbracket_g = \llbracket A \rrbracket_g, \llbracket B \rrbracket_g$ holds. We also assume that the built-ins present in the goal are left unchanged;
- Data sufficient (qualified) answers are preserved for the class of goals \mathcal{G} , that is, for all $G \in \mathcal{G} \subseteq \mathcal{G}_{CHR}$, $\mathcal{SA}_P(G) = \mathcal{SA}_{\llbracket P \rrbracket}(\llbracket G \rrbracket_g)$ ($\mathcal{QA}_P(G) = \mathcal{QA}_{\llbracket P \rrbracket}(\llbracket G \rrbracket_g)$) holds.

Separation Result

Theorem

Let \mathcal{G} be a class of goals such that if H is an head of a rule then $H \in \mathcal{G}$. When considering data sufficient or qualified answers, there exists no acceptable encoding of CHR in CHR-s for the class \mathcal{G} .

Hint on the proof

- Any possible encoding of the rule $r @ H, G \Leftrightarrow \text{true} \mid c$ into CHR-s would either produce more answers for the goal H (or G), or would not be able to provide the answer c for the goal H, G .

Testing the theorem

Program: Less than or equal \leq

reflexivity @ $LEQ(X, Y) \Leftrightarrow X = Y \mid \text{true}$

antisymmetry @ $LEQ(X, Y), LEQ(Y, X) \Leftrightarrow \text{true} \mid X = Y$

transitivity @ $LEQ(X, Y), LEQ(Y, Z) \Rightarrow \text{true} \mid LEQ(X, Z)$

- If a single headed program P is any translation of the program LEQ which produces the correct answer for the goal $LEQ(A, B), LEQ(B, C), LEQ(C, A)$,
- then there exists a subgoal which has an answer in P but not in the original program LEQ .

Summarizing...

- Multiple heads augment the expressive power of the language
- CHR cannot be encoded in CHR with single heads under quite reasonable assumptions

How about Prolog? (1)

Theorem

Let \mathcal{G} be a class of goals such that if H is an head of a rule then $H \in \mathcal{G}$. When considering data sufficient answers or qualified answers there exists no acceptable encoding of CHR in constraint logic programming nor in pure Prolog for the class \mathcal{G} .

Remarks

- CHR implementations usually work on top of Prolog: i.e. CHR is compiled into Prolog

How about Prolog? (2)

Remarks

- First, real Prolog systems use several non logical built-in's, which are out of the scope of our results

- It could also mean that, when considering pure Prolog, these compilers do not satisfy our assumptions (typically, they do not translate goals in a compositional way)

Future work

- we are working on weaker notions of acceptable encodings
- we are considering also the different expressive power of CHR_n , the language with at most n atoms in the heads, and CHR_{n+1}
- we are also planning to investigate what happens when considering translation of CHR into real Prolog systems (with non logical built-ins).

Example - the Sieve of Eratosthenes

Example

1 @ $upto(1) \Leftrightarrow \text{true}$

2 @ $upto(N) \Leftrightarrow N > 1 \mid \text{prime}(N), upto(N - 1)$

3 @ $\text{prime}(X), \text{prime}(Y) \Leftrightarrow X \bmod Y = 0 \mid \text{prime}(Y)$

Computation

- Let the goal be: $G = \{upto(5)\}$,

Example - the Sieve of Eratosthenes

Example

1 @ $upto(1) \Leftrightarrow \text{true}$

2 @ $upto(N) \Leftrightarrow N > 1 \mid prime(N), upto(N - 1)$

3 @ $prime(X), prime(Y) \Leftrightarrow X \bmod Y = 0 \mid prime(Y)$

Computation

- Let the goal be: $G = \{upto(5)\}$,
- after 4 applications of rule 2 the store contains $prime(2), prime(3), prime(4), prime(5)$

Example - the Sieve of Eratosthenes

Example

1 @ $upto(1) \Leftrightarrow \text{true}$

2 @ $upto(N) \Leftrightarrow N > 1 \mid \text{prime}(N), upto(N - 1)$

3 @ $\text{prime}(X), \text{prime}(Y) \Leftrightarrow X \bmod Y = 0 \mid \text{prime}(Y)$

Computation

- Let the goal be: $G = \{upto(5)\}$,
- after 4 applications of rule 2 the store contains $\text{prime}(2), \text{prime}(3), \text{prime}(4), \text{prime}(5)$
- then apply rule 3 to $\text{prime}(4), \text{prime}(2)$ hence remove $\text{prime}(4)$.

Example - the Sieve of Eratosthenes

Example

1 @ $upto(1) \Leftrightarrow \text{true}$

2 @ $upto(N) \Leftrightarrow N > 1 \mid \text{prime}(N), upto(N - 1)$

3 @ $\text{prime}(X), \text{prime}(Y) \Leftrightarrow X \bmod Y = 0 \mid \text{prime}(Y)$

Computation

- Let the goal be: $G = \{upto(5)\}$,
- after 4 applications of rule 2 the store contains $\text{prime}(2), \text{prime}(3), \text{prime}(4), \text{prime}(5)$
- then apply rule 3 to $\text{prime}(4), \text{prime}(2)$ hence remove $\text{prime}(4)$.
- no other rules are applicable

Example - the Sieve of Eratosthenes

Example

1 @ $upto(1) \Leftrightarrow \text{true}$

2 @ $upto(N) \Leftrightarrow N > 1 \mid \text{prime}(N), upto(N - 1)$

3 @ $\text{prime}(X), \text{prime}(Y) \Leftrightarrow X \bmod Y = 0 \mid \text{prime}(Y)$

Computation

- Let the goal be: $G = \{upto(5)\}$,
- after 4 applications of rule 2 the store contains $\text{prime}(2), \text{prime}(3), \text{prime}(4), \text{prime}(5)$
- then apply rule 3 to $\text{prime}(4), \text{prime}(2)$ hence remove $\text{prime}(4)$.
- no other rules are applicable
- the Qualified Answer obtained is $\{\text{prime}(2), \text{prime}(3), \text{prime}(5)\}$

Operational Semantics - formal

Solve	$\frac{CT \models c \wedge d \leftrightarrow d' \text{ and } c \text{ is a built-in constraint}}{\langle (c, G), K, d \rangle \longrightarrow \langle G, K, d' \rangle}$
Intro	$\frac{h \text{ is a user-defined constraint}}{\langle (h, G), K, d \rangle \longrightarrow \langle G, (h, K), d \rangle}$
Simp	$\frac{H \Leftrightarrow C \mid B \in P \quad x = Fv(H) \quad CT \models d \rightarrow \exists_x((H = H') \wedge C)}{\langle G, H' \wedge K, d \rangle \longrightarrow \langle B \wedge G, K, H = H' \wedge d \rangle}$
Prop	$\frac{H \Rightarrow C \mid B \in P \quad x = Fv(H) \quad CT \models d \rightarrow \exists_x((H = H') \wedge C)}{\langle G, H' \wedge K, d \rangle \longrightarrow \langle B \wedge G, H' \wedge K, H = H' \wedge d \rangle}$

Minsky machine encoding

$$\llbracket p_i : \text{Halt} \rrbracket_2 := i(p_i, R_1, R_2, X) \Leftrightarrow X = R_1$$

$$\llbracket p_i : \text{Succ}(r_1) \rrbracket_2 :=$$

$$i(p_i, R_1, R_2, X) \Leftrightarrow s(R_1, \text{Succ}R_1), i(p_{i+1}, \text{Succ}R_1, R_2, X)$$

$$\llbracket p_i : \text{DecJump}(r_1, p_l) \rrbracket_2 :=$$

$$i(p_i, R_1, R_2, X), s(\text{Pre}R_1, R_1) \Leftrightarrow i(p_{i+1}, \text{Pre}R_1, R_2, X)$$

$$\text{zero}(R_1), i(p_i, R_1, R_2, X) \Leftrightarrow i(p_l, R_1, R_2, X), \text{zero}(R_1)$$