

Analysis and Optimization of CHR Programs

ICLP 2005 Doctoral Consortium presentation

Jon Sneyers

jon@cs.kuleuven.be

Dept. Computer Science, K.U.Leuven, Belgium

Overview

1. Constraint Handling Rules
2. Guard and continuation optimization
3. Computability and Complexity
4. Ongoing work

1. Constraint Handling Rules

(Thom Frühwirth)

- Write your own constraint solver as CHRs:
 - ◆ application tailored solvers
 - ◆ embedded in Prolog (or other host language)
 - ⇒ no interfacing problems
 - ◆ high-level specification
 - focus on what, not how
 - fixpoint computation is taken care of
 - very compact programs
 - easy to understand, modify and experiment with
- Also useful as general-purpose language...
- K.U.Leuven CHR system (for SWI-Prolog, XSB, hProlog)

1. Constraint Handling Rules

Three kinds of CHR rules:

- Simplification rules:

$\text{RemovedHeads} \Leftrightarrow \text{Guard} \mid \text{Body}.$

- Propagation rules:

$\text{KeptHeads} \Rightarrow \text{Guard} \mid \text{Body}.$

- Simpagation rules:

$\text{KeptHeads} \setminus \text{RemovedHeads} \Leftrightarrow \text{Guard} \mid \text{Body}.$

A rule can be applied if:

- head constraints are in the constraint store
- the guard is satisfied

1. CHR example: interval solver

$\text{: - constraints in/2.}$

$X \text{ in } A:B \iff A > B \mid \text{fail.}$

$X \text{ in } A:B \iff A =: B \mid X \text{ is } A.$

$X \text{ in } A:B, X \text{ in } C:D \iff A < B, C < D \mid X \text{ in } \max(A,C):\min(B,D).$

- First rule: X is in an empty interval $\iff \text{fail}$
- Second rule: X is in $[a, a]$ $\iff X$ is a
- Third rule: interval intersection

Overview

1. Constraint Handling Rules
2. **Guard and continuation optimization**
3. Computability and Complexity
4. Ongoing work

2. Guard Optimization

$X \text{ in } A:B \iff A > B \mid \text{fail.}$

$X \text{ in } A:B \iff A ::= B \mid X \text{ is } A.$

$X \text{ in } A:B, X \text{ in } C:D \iff A < B, C < D \mid X \text{ in } \max(A,C):\min(B,D).$

- When last rule is tried, first two rules did not fire (otherwise active constraint was removed)
 \Rightarrow guards of first two rules failed for both constraints

2. Guard Optimization

$X \text{ in } A:B \iff A > B \mid \text{fail.}$

$X \text{ in } A:B \iff A ::= B \mid X \text{ is } A.$

$X \text{ in } A:B, X \text{ in } C:D \iff A < B, C < D \mid X \text{ in } \max(A,C):\min(B,D).$

- When last rule is tried, first two rules did not fire (otherwise active constraint was removed)
 \Rightarrow guards of first two rules failed for both constraints
- negations of first guard: $A \leq B$ and $C \leq D$
negations of second guard: $A \neq B$ and $C \neq D$
 \Rightarrow this entails $A < B$ and $C < D$
 \Rightarrow guard of last rule can be simplified

2. Guard Optimization

$X \text{ in } A:B \iff A > B \mid \text{fail.}$

$X \text{ in } A:B \iff A ::= B \mid X \text{ is } A.$

$X \text{ in } A:B, X \text{ in } C:D \iff X \text{ in } \max(A,C):\min(B,D).$

- When last rule is tried, first two rules did not fire (otherwise active constraint was removed)
 \Rightarrow guards of first two rules failed for both constraints
- negations of first guard: $A = < B$ and $C = < D$
negations of second guard: $A = \backslash = B$ and $C = \backslash = D$
 \Rightarrow this entails $A < B$ and $C < D$
 \Rightarrow guard of last rule can be simplified

2. Continuation Optimization

- Idea: skip constraint occurrences that don't result in rule application
- Example:

$\text{fib}(0, M) \implies M = 1.$

$\text{fib}(1, M) \implies M = 1.$

$\text{fib}(N, M) \implies N > 1 \mid \text{fib}(N-1, M1), \text{fib}(N-2, M2), M \text{ is } M1+M2.$

If the first rule is applied, don't try the next rules

2. Type and mode information

- Type and mode information can be used to improve optimizations (e.g. use hash-table store for ground constraints)
- $\text{sum}([],S) \iff S=0.$
 $\text{sum}([X|Xs],S) \iff \text{sum}(Xs,T), S \text{ is } X+T.$
- Optional type/mode declarations:
 $\text{:- chr_type list}(T) \longrightarrow [] ; [T \mid \text{list}(T)].$
 $\text{:- constraints sum}(+\text{list}(\text{int}), ?\text{int}).$
- Mode: + for ground arguments, ? for unknown mode.
- Better generated code...

2. Effect of Optimizations

```
:-use_module(library(chr_runtime)). :-use_module(library(chr_hashtable_store)). 'attach_sum/2'([],_). 'attach_sum/2'([E|D],C):- (get_attr(E,user,B)->A=[C|B],put_attr(E,user,A);put_attr(E,user,[C])), 'attach_sum/2'(D,C). 'detach_sum/2'([],_). 'detach_sum/2'([E|D],C):- (get_attr(E,user,B)->chr_runtime:sbag_del_element(B,C,A), (A=[]->del_attr(E,user);put_attr(E,user,A));true), 'detach_sum/2'(D,C). '$indexed_variables'(C,B):-C=sum(A,_), term_variables(A,B). attach_increment([],_). attach_increment([F|E],D):-chr_runtime:not_locked(F), (get_attr(F,user,C)->sort(C,B), chr_runtime:merge_attributes(D,B,A), put_attr(F,user,A);put_attr(F,user,D)), attach_increment(E,D). attr_unify_hook(G,F):-sort(G,E), (var(F)->(get_attr(F,user,D)->true;D=[]), sort(D,C), chr_runtime:merge_attributes(E,C,B), put_attr(F,user,B), chr_runtime:run_suspensions(B);(compound(F)->term_variables(F,A), attach_increment(A,E);true), chr_runtime:run_suspensions(G)). activate_constraint(H,G,F,E):-arg(2,F,D), D=mutable(C), chr_runtime:update_mutable(active,D), (nonvar(E)->true;arg(4,F,B), B=mutable(A), E is A+1, chr_runtime:update_mutable(E,B)), (compound(C)->term_variables(C,G), chr_runtime:none_locked(G), H=yes;C==removed->chr_indexed_variables(F,G), H=yes;G=[], H=no). remove_constraint_internal(E,D,C):-arg(2,E,B), B=mutable(A), chr_runtime:update_mutable(removed,B), (compound(A)->D=[], C=no;A==removed->D=[], C=no;C=yes, chr_indexed_variables(E,D)). insert_constraint_internal(yes,J,I,H,G,F):-I=..[suspension,E,D,H,C,B,G|F], chr_indexed_variables(I,J), chr_runtime:none_locked(J), chr_runtime:create_mutable(active,D), chr_runtime:create_mutable(0,C), chr_runtime:create_mutable(A,B), chr_runtime:empty_history(A), chr_runtime:gen_id(E). chr_indexed_variables(C,B):-C=..[_,_,_,_,_,_,_|_], '$indexed_variables'(A,B). '$insert_in_store_sum/2'(D):-chr_runtime:global_term_ref_1(C), (get_attr(C,user,B)->A=[D|B], put_attr(C,user,A);put_attr(C,user,[D])). '$delete_from_store_sum/2'(D):-chr_runtime:global_term_ref_1(C), (get_attr(C,user,B)->chr_runtime:sbag_del_element(B,D,A), (A=[]->del_attr(C,user);put_attr(C,user,A));true). '$enumerate_suspensions'(C):-chr_runtime:global_term_ref_1(B), get_attr(B,user,A), chr_runtime:sbag_member(C,A). sum(B,A):-'sum/2__0'(B,A,_). 'sum/2__0'(E,D,C):-E=[]!, (var(C)->true;remove_constraint_internal(C,B,A), (A==yes->'$delete_from_store_sum/2'(C), 'detach_sum/2'(B,C);true)), D=0. 'sum/2__0'(H,G,F):-nonvar(H), H=[E|D]!, (var(F)->true;remove_constraint_internal(F,C,B), (B==yes->'$delete_from_store_sum/2'(F), 'detach_sum/2'(C,F);true)), sum(D,A), G is E+A. 'sum/2__0'(E,D,C):- (var(C)->insert_constraint_internal(B,A,C,user:'sum/2__0'(E,D,C), sum(E,D), [E,D])); activate_constraint(B,A,C,_), (B==yes->'$insert_in_store_sum/2'(C), 'attach_sum/2'(A,C);true).
```



```
:- use_module(library(chr_runtime)).  
:- use_module(library(chr_hashtable_store)).  
'$enumerate_suspensions'(_) :- fail.  
sum([],A) :- !, A=0.  
sum([D|C],B) :- sum(C,A), B is D+A.
```

2. Guard and continuation optimization

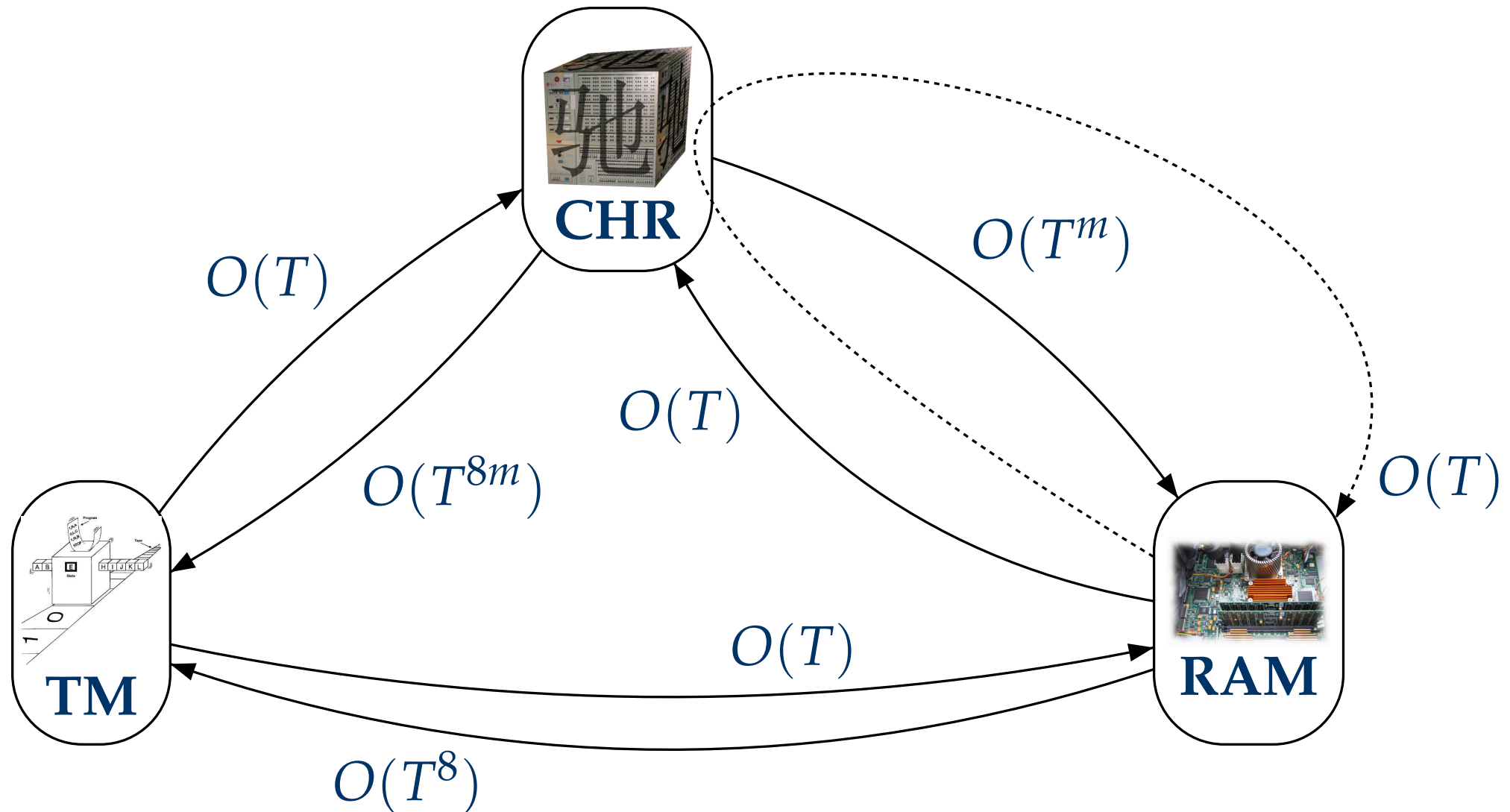
- More information:

- ◆ Jon Sneyers, Tom Schrijvers and Bart Demoen. *Guard simplification in CHR programs*. Proceedings of the 19th Workshop on (Constraint) Logic Programming, W(C)LP 2005, Ulm, Germany, February 2005.
- ◆ Jon Sneyers, Tom Schrijvers and Bart Demoen. *Guard and Continuation Optimization for Occurrence Representations of CHR*. 21st International Conference on Logic Programming (ICLP'05), Sitges, Spain, October 2005.

Overview

1. Constraint Handling Rules
2. Guard and continuation optimization
3. **Computability and Complexity**
4. Ongoing work

3. Computability and Complexity



3. Computability and Complexity

- CHR is Turing complete
- Every algorithm can be implemented in CHR with the best known time/space complexity!

3. Computability and Complexity

- More information:

- ◆ Jon Sneyers, Tom Schrijvers and Bart Demoen. *The Computational Power and Complexity of Constraint Handling Rules*. 2nd Workshop on Constraint Handling Rules (CHR'05) at ICLP'05, Sitges, Spain, October 2005. Presentation at the CHR Workshop:
Wednesday, 16h45, Room L1evant 2

Overview

1. Constraint Handling Rules
2. Guard and continuation optimization
3. Computability and Complexity
4. **Ongoing work**

4. Ongoing work

- What about the constant factor / elegance of programs?
- Implement classic algorithms in CHR
- Dijkstra's shortest path algorithm using a Fibonacci heap has an elegant CHR implementation
“executable specification”
- For sufficiently optimizing CHR systems it has the optimal time complexity of $O(n \log n)$
- Some preliminary results....

Single-source shortest path on sparse positive weight directed graphs using different implementations of the Dijkstra algorithm

