

Guard Simplification in CHR programs

WCLP 2005
Ulm, Germany

Jon Sneyers, Tom Schrijvers, Bart Demoen
{jon,toms,bmd}@cs.kuleuven.ac.be.

K.U.Leuven, Belgium

Overview

- 1. Constraint Handling Rules (CHR)
- 2. ω_t and ω_r semantics
- 3. Guard simplification
 - ◆ 3.1 Basic idea
 - ◆ 3.2 Head matching simplification
 - ◆ 3.3 Type and mode declarations
- 4. Implementation
- 5. Experimental results
- 6. Conclusion & Future 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 a general-purpose language

1. Constraint Handling Rules

Three kinds of CHR rules:

- Simplification rules:

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

- Propagation rules:

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

- Simplification rules:

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

1. CHR: Example: interval solver

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

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

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

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

1. CHR: Example: interval solver

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

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

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

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

Prompt:

?- V in 3:8, V in 2:6, V in 0:3.

1. CHR: Example: interval solver

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

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

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

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

Prompt:

?- $V \text{ in } 3:8, V \text{ in } 2:6, V \text{ in } 0:3.$

Constraint store:

$V \text{ in } 3:8$

1. CHR: Example: interval solver

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

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

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

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

Prompt:

?- V in 3:8, V in 2:6, V in 0:3.

Constraint store:

V in 3:8

V in 2:6

1. CHR: Example: interval solver

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

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

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

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

Prompt:

?- V in 3:8, V in 2:6, V in 0:3.

Constraint store:

V in 3:8

V in 2:6

1. CHR: Example: interval solver

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

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

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

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

Prompt:

?- $V \text{ in } 3:8, V \text{ in } 2:6, V \text{ in } 0:3.$

Constraint store:

$V \text{ in } 3:6$

1. CHR: Example: interval solver

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

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

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

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

Prompt:

?- V in 3:8, V in 2:6, V in 0:3.

Constraint store:

V in 3:6

V in 0:3

1. CHR: Example: interval solver

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

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

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

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

Prompt:

?- $V \text{ in } 3:8, V \text{ in } 2:6, V \text{ in } 0:3.$

Constraint store:

$V \text{ in } 3:6$

$V \text{ in } 0:3$

1. CHR: Example: interval solver

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

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

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

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

Prompt:

?- V in 3:8, V in 2:6, V in 0:3.

Constraint store:

V in 3:3

1. CHR: Example: interval solver

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

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

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

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

Prompt:

?- $V \text{ in } 3:8, V \text{ in } 2:6, V \text{ in } 0:3.$

Constraint store:

$V \text{ in } 3:3$

1. CHR: Example: interval solver

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

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

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

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

Prompt:

?- $V \text{ in } 3:8, V \text{ in } 2:6, V \text{ in } 0:3.$

$V = 3$

Yes

Constraint store:

2. Operational semantics

- theoretical operational semantics ω_t
 - ◆ rules can be applied in any order
 - ◆ confluence is nontrivial
- refined operational semantics ω_r
 - ◆ an instance of ω_t
 - ◆ rules applied in textual order
 - ◆ better termination/complexity possible
 - ◆ used in all major CHR implementations

2. Operational semantics

Most CHR programmers use the refined operational semantics (ω_r).

	ω_t	ω_r
clear logical reading	✓	✗
easy to implement things like key lookup	✗	✓
efficient compiled code	✗	✓
programs are correct under other semantics	✓	✗

3.1 Guard Simplification: Basic Idea

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

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

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

- 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

3.1 Guard Simplification: Basic Idea

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

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

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

- 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

3.1 Guard Simplification: Basic Idea

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

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

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

- 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: $A \leq B$ 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

3.1 Guard Simplification: Basic Idea

- Order of rules is implicit precondition
- Programmers don't write guards entailed by rule order
 - ✓ more compact and efficient programs
 - ✗ logical reading obfuscated

3.1 Guard Simplification: Basic Idea

- Order of rules is implicit precondition
- Programmers don't write guards entailed by rule order
 - ✓ more compact and efficient programs
 - ✗ logical reading obfuscated
- Solution: Guard Simplification
 - ◆ Automatically remove entailed guards
 - ◆ Programmer can put all preconditions in guard
 - ⇒ logical reading restored
- Advantages of both ω_t and ω_r !

3.1 GS: Basic Idea: in general

$$KH_1^1, KH_2^1, \dots, KH_{k_1}^1 \setminus RH_1^1, RH_2^1, \dots, RH_{r_1}^1 \iff G^1 \mid B^1.$$

$$KH_1^2, KH_2^2, \dots, KH_{k_2}^2 \setminus RH_1^2, RH_2^2, \dots, RH_{r_2}^2 \iff G^2 \mid B^2.$$

⋮

$$\underbrace{KH_1^i, KH_2^i, \dots, KH_{k_i}^i \setminus RH_1^i, RH_2^i, \dots, RH_{r_i}^i}_{H^i} \iff G^i \mid B^i.$$

⋮

$$KH_1^n, KH_2^n, \dots, KH_{k_n}^n \setminus RH_1^n, RH_2^n, \dots, RH_{r_n}^n \iff G^n \mid B^n.$$

The guard of the i -th rule can be removed if:

$$\bigwedge \left\{ \neg G^j \mid \underbrace{(1 \leq j < i) \wedge (H^j \subseteq H^i)}_{\text{rule } j \text{ is an earlier subrule of rule } i} \wedge (r_j > 0) \right\} \rightarrow G^i$$

3.2 Head matching simplification

$p(X, Y) \iff X \text{ _== } Y \mid \text{Body1}$

$p(\color{red}{X}, \color{red}{X}) \iff \text{Body2}$

- Head matchings are implicit part of guard

3.2 Head matching simplification

$p(X, Y) \Leftrightarrow X \backslash == Y \mid \text{Body1}$

$p(X, Y) \Leftrightarrow X == Y \mid \text{Body2}$

- Head matchings are implicit part of guard
⇒ make them explicit and do guard simpl.

3.2 Head matching simplification

$p(X, Y) \Leftrightarrow X \backslash == Y \mid \text{Body1}$

$p(X, Y) \Leftrightarrow \text{Body2}$

- Head matchings are implicit part of guard
 \Rightarrow make them explicit and do guard simpl.
- More general head
- Easier to detect $p / 2$ is a never-stored constraint

3.3 Type and mode declarations

$\text{sum}([], S) \Leftrightarrow S = 0.$

$\text{sum}([X|Xs], S) \Leftrightarrow \text{sum}(Xs, S2), S \text{ is } X+S2.$

- $\text{sum}/2$ is never-stored, but we can't detect it

3.3 Type and mode declarations

$\text{- chr_type list}(T) \longrightarrow [] ; [T \mid \text{list}(T)].$

$\text{- constraints sum}(+\text{list}(\text{int}), ?\text{int}).$

$\text{sum}([], S) \iff S = 0.$

$\text{sum}([X \mid Xs], S) \iff \text{sum}(Xs, S2), S \text{ is } X+S2.$

- $\text{sum}/2$ is never-stored, but we can't detect it
 \Rightarrow introduce type and mode declarations

3.3 Type and mode declarations

$\text{- chr_type list}(T) \longrightarrow [] ; [T \mid \text{list}(T)].$

$\text{- constraints sum}(+\text{list}(\text{int}), ?\text{int}).$

$\text{sum}([], S) \iff S = 0.$

$\text{sum}([X|Xs], S) \iff \text{sum}(Xs, S2), S \text{ is } X+S2.$

- $\text{sum}/2$ is never-stored, but we can't detect it
 - \Rightarrow introduce type and mode declarations
 - \Rightarrow head matching can be simplified
- in this case, moved to the body because rhs vars are needed

3.3 Type and mode declarations

$\text{- chr_type list}(T) \longrightarrow [] ; [T \mid \text{list}(T)].$

$\text{- constraints sum}(+\text{list}(\text{int}), ?\text{int}).$

$\text{sum}([], S) \iff S = 0.$

$\text{sum}(A, S) \iff A = [X \mid Xs], \text{sum}(Xs, S2), S \text{ is } X + S2.$

- $\text{sum}/2$ is never-stored, but we can't detect it
 - \Rightarrow introduce type and mode declarations
 - \Rightarrow head matching can be simplified
- in this case, moved to the body because rhs vars are needed
- \Rightarrow easy to detect never-stored property

4. Implementation

- Implemented in K.U.Leuven CHR compiler
 - ◆ both in hProlog+CHR and SWI-Prolog+CHR
 - ◆ added support for type defs/decls
- Implemented as new compilation phase:
 - ◆ source to source transformation
 - ◆ derive information from rule order, types, modes
 - ◆ if guard is entailed, remove guard
 - ◆ if \neg guard is entailed, give warning & remove rule
- Entailment check module (written in CHR)

4. Implementation: generated code

Example CHR program:

```
filter([X|In],P,Out) <=> 0 =\= X mod P |  
                          Out=[X|Out1],  
                          filter(In,P,Out1).
```

```
filter([X|In],P,Out) <=> 0 ::= X mod P |  
                          filter(In,P,Out).
```

```
filter([],P,Out) <=> Out=[].
```

4. Implementation: generated code

Without guard simplification:

```
filter(List,P,Out) :- filter(List,P,Out, _ ).
```

```
filter(List,P,Out,C) :-                % first occurrence
    nonvar(List), List = [X|In], 0 =\= X mod P, !,
    ... % remove from constraint store if needed
    Out = [E|Out1], filter(In,P,Out1).
```

```
filter(List,P,Out,C) :-                % second occurrence
    nonvar(List), List = [X|In], 0 ::= X mod P, !,
    ... % remove from constraint store if needed
    filter(In,P,Out).
```

4. Implementation: generated code

Without guard simplification: (continued)

```
filter(List, _ ,Out,C) :-                % third occurrence
    List == [], !,
    ... % remove from constraint store if needed
    Out = [].
```

```
% insert into store in case none of the rules matched
filter(List,P,Out,C) :-
    ... % insert into constraint store
```

4. Implementation: generated code

With guard simplification (and type/mode info):

```
filter(List,P,Out) :- List = [X|In], 0 =\= X mod E, !,  
                    Out = [X|Out1], filter(In,P,Out1).
```

```
filter(List,P,Out) :- List = [_|In], !, filter(In,P,Out).
```

```
filter(_,_ ,Out) :- Out = [].
```

5. Experimental results

<i>Benchmark</i>	<i>Lang</i>	<i>GS</i>	<i>Mode</i>	<i>Type</i>	<i>Codesize</i>	<i>Runtime</i>
sum (10000,500)		—	×	×	4 , 46	100.0%
	CHR	—	✓	×	3 , 10	88.9%
		✓	✓	✓	2 , 6	66.7%
	handwritten Prolog code				2 , 5	66.1%
Takeuchi (1000)		×	×	—	4 , 50	100.0%
	CHR	×	✓	—	3 , 17	65.8%
		✓	—	—	2 , 12	61.0%
	handwritten Prolog code				2 , 12	61.0%
nrev (30,50000)		—	×	×	8 , 92	100.0%
	CHR	—	✓	×	6 , 20	62.9%
		✓	✓	✓	4 , 11	23.0%
	handwritten Prolog code				4 , 7	20.5%

5. Experimental results

<i>Benchmark</i>	<i>Lang</i>	<i>GS</i>	<i>Mode</i>	<i>Type</i>	<i>Codesize</i>	<i>Runtime</i>
cprimes (100000)	CHR	×	×	—	14 , 160	100.0%
		×	✓	—	11 , 42	58.1%
		✓	×	×	12 , 120	99.4%
		✓	✓	×	10 , 35	57.2%
		✓	✓	✓	8 , 25	55.8%
	handwritten Prolog code				8 , 23	55.8%
dfsearch (16,500)	CHR	×	×	—	5 , 67	100.0%
		×	✓	—	4 , 16	84.4%
		✓	×	×	5 , 66	90.7%
		✓	✓	×	4 , 15	79.4%
		✓	✓	✓	3 , 11	59.5%
	handwritten Prolog code				3 , 8	55.8%

5. Experimental results

- Generated code much closer to Prolog
- CHR programmers usually write deterministic parts in Prolog for efficiency
⇒ can write everything in CHR now
- Constraint not never-stored → only minor improvement (guards are usually cheap)
- Never-stored constraint → big improvement

5. Experimental results

- Performance of guard simplification phase itself: reasonable
- Scales well ...

- Compiling K.U.Leuven CHR compiler: (139 rules, 73 constraints) — 3.3 seconds, 1.5 for GS

5. Experimental results

- Performance of guard simplification phase itself: reasonable
- Scales well ... **except when there are many rules and few constraints!**
 - complexity depends on number of earlier subrules
- Compiling K.U.Leuven CHR compiler:
(139 rules, 73 constraints) — 3.3 seconds, 1.5 for GS
- Compiling entailment checker:
(123 rules, 3 constraints) — 2.1 seconds, **1.8 for GS !**

5. Experimental results

<i>program</i>	<i>R</i>	<i>C</i>	<i>R/C</i>	<i>GS</i>	<i>TC</i>	<i>GS/TC</i>
union-find	6	6	1.0	0	40	0.0%
timed automata	23	17	1.4	50	310	16.1%
well-founded semantics	43	18	2.4	70	340	20.6%
finite domain solver	13	6	2.2	80	200	40.0%
CHR compiler	139	73	1.9	1,540	3,270	47.1%
boolean solver	78	8	9.8	420	590	71.2%
(in)finite domain solver	81	9	9.0	950	1,250	76.0%
entailment checker	123	3	41.0	1,830	2,150	85.1%

GS: Guard simplification phase time (ms)

TC: Total compilation time (ms)

R: number of rules

C: number of constraints

6. Conclusion

- GS allows more declarative CHR programs
- Allows other analyses to detect more cases (e.g. never-stored analysis)
- Type and mode information → more simplification
- Generated code is as close to Prolog as possible
 - ◆ more efficient
 - ◆ no need to write parts in Prolog for efficiency

6. Future work

Current work:

- Improve scalability
- Occurrence subsumption: generalizes symmetry

analysis:

$c(X) \setminus c(X) \Leftrightarrow \dots$

$c(X, Y), c(Y, X) \Leftrightarrow \text{true} \mid \dots$

$c(X, Y, Z), c(Y, Z, X), c(Z, X, Y) \Leftrightarrow (p(X); p(Y)) \mid \dots$

6. Future work

Future work:

- Support for declarations of properties of user-defined predicates
- Use info derived for GS in other phases
- Program specialization on constraint calls in bodies based on derived properties of call arguments

Questions?

Questions?

Related papers (same authors):

- *Guard Simplification in CHR programs*. Technical Report CW 396, K.U.Leuven, Department of Computer Science, November 2004.
- *Guard Reasoning for CHR Optimization*. Submitted to PPDP 2005, Lisboa, Portugal, July 2005.