

# The K.U.Leuven CHR System: Implementation and Application

Tom Schrijvers, Bart Demoen

{tom.schrijvers,bart.demoen}@cs.kuleuven.ac.be.



Katholieke Universiteit Leuven, Belgium

# Overview of the Talk

- 1. Implementation
  - ◆ 1.1 Setup and Evolution
  - ◆ 1.2 Optimizations
  - ◆ 1.3 Evaluation
- 2. Application
  - ◆ 2.1 Well-Founded Semantics
  - ◆ 2.2 Java Memory Model
- 3. Conclusion
  - ◆ 3.1 Future Work
  - ◆ 3.2 Discussion Topics



# 1. Implementation

# 1.1 Setup and Evolution

K.U.Leuven CHR system consists of two parts:

- **the runtime**
  - ◆ strongly based on Christian Holzbaaur's
  - ◆ finetuned for performance
- **the preprocessing compiler**
  - ◆ CHR  $\implies$  Prolog

# 1.1 Setup and Evolution

- hProlog (Bart Demoen)
  - ◆ evaluate new attributed variables
  - ◆ optimized compilation of CHRs
- XSB (David S. Warren)
  - ◆ along with hProlog's attributed variables
  - ◆ integration of CHR with tabling (ICLP'04)
- SWI-Prolog (Jan Wielemaker)
  - ◆ along with hProlog's attributed variables
  - ◆ first CLP capabilities in SWI-Prolog

# 1.2 Optimizations

Based on other systems and own ideas:

- heuristic reordering of constraints in multi-headed rules
- early scheduling of guards
- detection of never attached constraints
  - ⇒ other constraints in same rule passive
  - ⇒ avoid space and time overhead

# 1.2 Optimizations

- functional dependencies inference  
⇒ *once* transformation

```
a(X,_) \ a(X,_) <=> true.
```

```
b(X), ..., a(X,Y), ... ==> ...
```

```
procedure_b(X) :-
```

```
    ... % retrieve other constraints
```

```
    once(
```

```
        retrieve_a(X,Constraint)
```

```
    ),
```

```
    ... % retrieve other constraints
```

```
    !,
```

```
    ...
```

# 1.2 Optimizations

- inference of unmatched arguments  
⇒ no attaching

```
entry(Key, Value) \ lookup(Key, Query)  
=> Query = Value.
```

```
?- entry(123, [Var1, Var2, ..., Var10]),  
   lookup(123, Query).
```

`entry/2` is redundantly (de/at)tached

# 1.2 Optimizations

- inference of unmatched arguments conditions for argument in all rules:
  - ◆ argument is a variable
  - ◆ variable only appears once in heads and guard
  - ◆ variable may appear in (anti-monotonic) var test:

`fib(N,M1) \ fib(N,M2) <=> var(M2) | M1 = M2.`

`fib(N,M) ==> N =< 1 | M = 1.`

`fib(N,M) ==> N > 1 | N1 is N-1, fib(N1,M1),  
N2 is N-2, fib(N2,M2), M is M1 + M2.`

# 1.3 Evaluation

Factors:

- Plain Prolog performance

	SICStus	Yap	hProlog	XSB	SWI-Prolog
average	100.0%	78.0%	76.2%	146.8%	448.2%

- CHR optimizations + instantiation errors (disabled)
- Attributed variables implementation
  - ◆ global store is hotspot
  - ◆ short reference chains

# 1.3 Evaluation

Benchmark	Christian Holzbaur		K.U.Leuven		
	SICStus	Yap	hProlog	XSB	SWI-Prolog
bool	100.0%	74.1%	43.3%	86.0%	207.0%
fib	100.0%	61.7%	76.5%	154.9%	538.9%
fibonacci	100.0%	59.6%	35.4%	82.2%	270.0%
leq	100.0%	96.0%	81.6%	151.1%	454.4%
primes	100.0%	104.8%	51.3%	139.5%	520.3%
ta	100.0%	82.6%	53.1%	106.2%	380.4%
wfs	100.0%	63.6%	52.5%	125.9%	309.3%
zebra	100.0%	52.6%	21.3%	50.8%	139.1%
average	100.0%	74.4%	51.9%	112.1%	352.4%



## 2. Application

# 2.1 Well-Founded Semantics

Algorithm to compute WFS of logic programs without arguments. Fixpoint of 2 steps:

- step 1
  - ◆ decide truth of atoms based on facts and already defined atoms
- step2
  - ◆ only consider undefined atoms and positive body literals
  - ◆ apply same reasoning as in step 1
  - ◆ still undefined: false are considered false
  - ◆ now defined: remain undefined

# 2.1 Well-Founded Semantics

Prototype implementation in CHR

- sequencing of steps:  
flag constraints to enable/disable rules
- stop fixpoint:  
dirty flag constraint + order of rules
- different semantics step 1 vs. step 2:  
replace  $c1 / n$  with  $c2 / n$
- available as `wfs.chr`

Refined operational semantics crucial!

# 2.1 Well-Founded Semantics

Some used CHR idioms/patterns/hacks:

```
witness2 \ witness2 <=> true.  
phase2, nbucl(At,_) ==> witness2, undefined2(At).  
phase2, pos(At,C1) ==> pos2(At,C1).  
phase2, aclause(C1,At) ==> aclause2(C1,At).  
phase2, nbplit(C1,N) ==> nbplit2(C1,N).  
phase2, witness2 # ID <=> phase1 pragma passive(ID).  
phase2 \ nbplit2(_,_) # ID <=> true pragma passive(ID).  
phase2 \ aclause2(_,_) # ID <=> true pragma passive(ID).  
phase2 <=> true.
```

## 2.2 Java Memory Model

Java Memory Model flawed

- memory model: interaction threads - main memory
- JSR-133: new memory model
- Concurrent Constraint-based Memory Machines (V. Saraswat):
  - ◆ framework for memory models
  - ◆ events + constraints
  - ◆ rules for ordering between events
  - ◆ rules for linking reads to writes

## 2.2 Java Memory Model

### CHR implementation

- prove claim of CCMMs: generative models
- CHR used for
  - ◆ finite domain constraints
  - ◆ partial order
  - ◆ ordering and linking rules
- poster at ICLP'04

A decorative graphic consisting of a light green vertical bar on the left side of the slide, a light green horizontal bar at the top left corner, and a thick dark blue horizontal bar extending across the top of the slide.

# 3. Conclusion

# 3. Conclusion

- fairly portable CHR system
  - ◆ hProlog
  - ◆ XSB
  - ◆ SWI-Prolog
- competitive performance
- tabling in XSB

# 3.1 Future Work

- Debugger (Jan Wielemaker)
- Optimizations
  - ◆ Instantiation and other declarations
  - ◆ Intelligent backtracking (+ backmarking)?
  - ◆ CHR without global store?
- Tabling specific features and applications in XSB

# 3.1 Discussion Topics

- One CHR Standard
  - ◆ Syntax
  - ◆ Semantics
- Portable Options and Pragmas
- Programming in the large?
- Benchmark Suite