

Analyses, Optimizations and Extensions of Constraint Handling Rules

Tom Schrijvers

June 10, 2005





1 Introduction

- Constraint Handling Rules
- Illustration: Show Cases of CHR
- Context
- Goals

2 Contributions

- The K.U.Leuven CHR System
- Abstract Interpretation
- CHR and Tabled Execution
- Automatic Implication Checking

3 Conclusion

- Future Work on CHR



1 Introduction

- Constraint Handling Rules
- Illustration: Show Cases of CHR
- Context
- Goals

2 Contributions

- The K.U.Leuven CHR System
- Abstract Interpretation
- CHR and Tabled Execution
- Automatic Implication Checking

3 Conclusion

- Future Work on CHR

Introduction to Constraint Handling Rules (CHR)

5/34

What is CHR (Thom Frühwirth)?

- ▶ input: constraints (e.g. $X \leq Y$)
- ▶ program: conditional rewriting rules
- ▶ output: solved form, after exhaustive rule application
- ▶ how: don't worry (unspecified)

What are the advantages of CHR?

- ▶ **what**, not **how**
- ▶ compact programs
- ▶ easy to extend and modify
- ▶ practical: **how** when necessary

An Example of CHR

6/34

A Less-Than-Or-Equal-To Constraint

```

refl @ X ≤ X                <=> true.
antisym @ X ≤ Y , Y ≤ X    <=> X = Y.
redund @ X ≤ Y \ X ≤ Y    <=> true.
trans @ X ≤ Y , Y ≤ Z     ==> X ≤ Z.
  
```

An Example of CHR

6/34

A Less-Than-Or-Equal-To Constraint

```

refl @ X ≤ X                <=> true.
antisym @ X ≤ Y , Y ≤ X    <=> X = Y.
redund @ X ≤ Y \ X ≤ Y    <=> true.
trans @ X ≤ Y , Y ≤ Z     ==> X ≤ Z.
  
```

A Query

```
?- X ≤ Y, Y ≤ Z.
```

An Example of CHR

6/34

A Less-Than-Or-Equal-To Constraint

```

refl @ X ≤ X                <=> true.
antisym @ X ≤ Y , Y ≤ X    <=> X = Y.
redund @ X ≤ Y \ X ≤ Y    <=> true.
trans @ X ≤ Y , Y ≤ Z     ==> X ≤ Z.
  
```

A Query

```
?- X ≤ Y, Y ≤ Z.
```

The Constraint Store

An Example of CHR

6/34

A Less-Than-Or-Equal-To Constraint

```

refl @ X ≤ X                <=> true.
antisym @ X ≤ Y , Y ≤ X    <=> X = Y.
redund @ X ≤ Y \ X ≤ Y    <=> true.
trans @ X ≤ Y , Y ≤ Z     ==> X ≤ Z.
  
```

A Query

```
?-          Y ≤ Z.
```

The Constraint Store

```
X ≤ Y
```

An Example of CHR

6/34

A Less-Than-Or-Equal-To Constraint

```

refl @ X ≤ X                <=> true.
antisym @ X ≤ Y , Y ≤ X    <=> X = Y.
redund @ X ≤ Y \ X ≤ Y    <=> true.
trans @ X ≤ Y , Y ≤ Z     ==> X ≤ Z.
  
```

A Query

```
?- .
```

The Constraint Store

```
X ≤ Y, Y ≤ Z
```

An Example of CHR

6/34

A Less-Than-Or-Equal-To Constraint

```

refl @ X ≤ X                <=> true.
antisym @ X ≤ Y , Y ≤ X    <=> X = Y.
redund @ X ≤ Y \ X ≤ Y    <=> true.
trans @ X ≤ Y , Y ≤ Z     ==> X ≤ Z.
    
```

A Query

```
?- .
```

The Constraint Store

```
X ≤ Y, Y ≤ Z
```

An Example of CHR

6/34

A Less-Than-Or-Equal-To Constraint

```

refl @ X ≤ X                <=> true.
antisym @ X ≤ Y , Y ≤ X    <=> X = Y.
redund @ X ≤ Y \ X ≤ Y    <=> true.
trans @ X ≤ Y , Y ≤ Z     ==> X ≤ Z.
  
```

A Query

```
?- .
```

The Constraint Store

```
X ≤ Y, Y ≤ Z, X ≤ Z
```

An Example of CHR

6/34

A Less-Than-Or-Equal-To Constraint

```

    refl @ X ≤ X                <=> true.
    antisym @ X ≤ Y , Y ≤ X    <=> X = Y.
    redund @ X ≤ Y \ X ≤ Y    <=> true.
    trans @ X ≤ Y , Y ≤ Z     ==> X ≤ Z.
  
```

A Query

```
?- .
```

The Constraint Store

```
X ≤ Y, Y ≤ Z, X ≤ Z
```

What is CHR for?

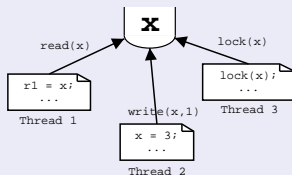
- ▶ custom constraint solvers
find solution to constraints
- ▶ typical application domains:
 - ▶ planning
 - ▶ scheduling
- ▶ new domains:
 - ▶ type inference and checking
 - ▶ natural language processing
 - ▶ multi-agent systems
- ▶ any general purpose application

Show Cases of CHR

8/34

- 1 Java Memory Model
- 2 Well-Founded Semantics
- 3 The Union-Find Algorithm

Java Memory Model (JMM)



memory model:
 specification of
 memory \leftrightarrow
 threads interaction

old JMM:
 “/tmp” \rightarrow “/dev”
new JMM: more
 intuitive

CHR Implementation: JMM^{SOLVE}

Java Memory Model (JMM)

CHR Implementation: JMM_{SOLVE}

CCMM declarative framework (V. Saraswat)
 in terms of rules and constraints

JMM_{SOLVE} implementation in Prolog and CHR
 eg. conditional write:

```
if (y > 0) x := 1; || r := x;
```

```
ifThenElse(true,X1,X2,R)  <=> R = X1.
```

```
ifThenElse(false,X1,X2,R) <=> R = X2.
```

```
ifThenElse(Cond,X,X,R)   <=> R = X.
```

Well-Founded Semantics \mathbf{W}_P^*

- ▶ general logic program P

ex. $a \leftarrow a, \neg a.$

- ▶ truth atom a :

true $a \in \mathbf{W}_P^*$

false $\neg a \in \mathbf{W}_P^*$

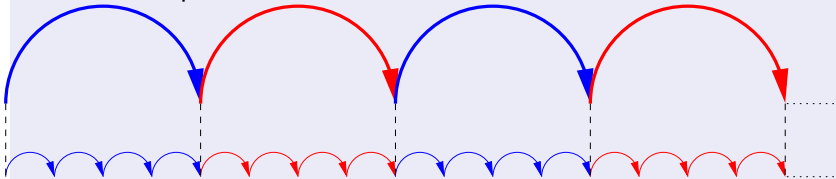
CHR Implementation

Well-Founded Semantics W_P^*

CHR Implementation

algorithm of P. Simons

- ▶ order important: low-level code



- ▶ confluent: high-level code

The Union-Find Algorithm

- ▶ classic algorithm
- ▶ many applications: logic variable, minimal spanning tree, ...
- ▶ many improvements: path compression, union-by-rank, ...
- ▶ best known time complexity: near-linear

CHR Implementation (with T. Frühwirth)

The Union-Find Algorithm

CHR Implementation (with T. Frühwirth)

- ▶ close to imperative formulation
- ▶ improvements: small changes
- ▶ time complexity:
 - ▶ general time complexity **insufficient**
 - ▶ operational equivalence
 - ▶ time complexity of CHR system
 - ▶ \Rightarrow near-linear complexity

Paradigms Related to CHR

12/34

Constraint Logic Programming

- ▶ CHR embeds well in CLP
- ▶ CHR implements solvers for CLP

Rule-Based Languages

- ▶ Production Rule Systems
- ▶ Term Rewriting Systems

Major Goals of this Thesis

- 1 Optimized Compilation: [The K.U.Leuven CHR System](#)
 - ▶ consolidation of state-of-the-art
 - ▶ new optimizations
- 2 Program Analysis: [Abstract Interpretation](#)
 - ▶ systematic approach
 - ▶ new analyses
- 3 Extensions of Expressivity
 - ▶ integration with CLP features
[tabled execution](#)
 - ▶ better support for constraint solvers
[implication checking](#)



- 1 Introduction
 - Constraint Handling Rules
 - Illustration: Show Cases of CHR
 - Context
 - Goals
- 2 Contributions
 - The K.U.Leuven CHR System
 - Abstract Interpretation
 - CHR and Tabled Execution
 - Automatic Implication Checking
- 3 Conclusion
 - Future Work on CHR

Issue: Dated CHR Implementations

- ▶ in Prolog, Haskell, Java
- ▶ reference implementation (Holzbaur&Frühwirth 1996)
- ▶ several obsolete: lost, incomplete, ...
- ▶ no consolidation of optimizations
- ▶ no more evolution?

Solution: The K.U.Leuven CHR System

- ▶ a state-of-the-art CHR system
- ▶ with both existing optimizations (Holzbaur, Duck, Stuckey)
- ▶ and **new** optimizations
- ▶ open source
- ▶ portable

New Optimizations

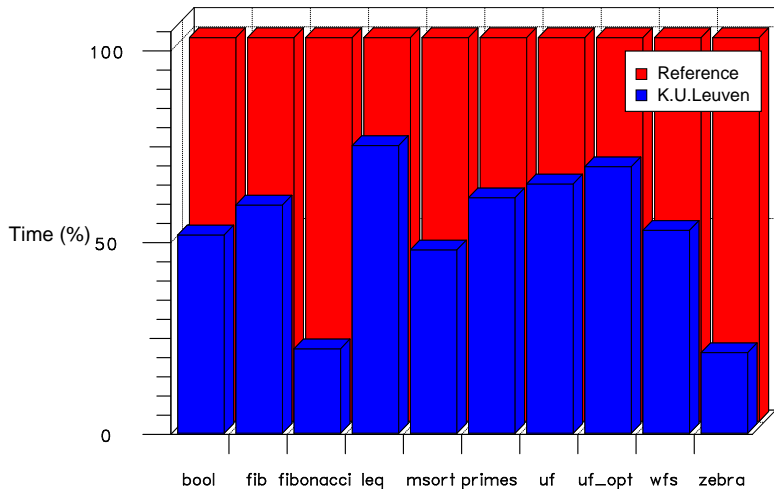
- ▶ anti-monotonic delay avoidance
- ▶ code specialization for ground constraints
- ▶ strong late storage optimization
- ▶ hashtable constraint stores

Performance

- ▶ very competitive performance
- ▶ sometimes better time complexity

K.U.Leuven CHR Performance

18/34



Benchmarks



Available in 3 Prolog Systems

19/34

- ▶ originally written for hProlog (B. Demoen)
- ▶ ported to XSB (D.S. Warren)



- ▶ ported to SWI-Prolog (J. Wielemaker)



Contribution 2: Abstract Interpretation of CHR

20/34

Problem

- ▶ program analyses drive optimized compilation
- ▶ several adhoc program analyses
- ▶ but not explained
- ▶ difficult to compare and improve

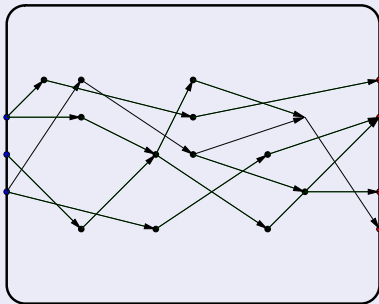
Solution : Abstract Interpretation (with G. Duck and P. Stuckey)

- ▶ established methodology
- ▶ formal description
- ▶ easy to compare, improve and combine analyses
- ▶ reach a higher level in analysis complexity
- ▶ language independent

Abstract Interpretation Approach

22/34

Concrete Derivations



concrete semantics:

- ▶ derivation steps
- ▶ from **initial state**
- ▶ to **final state**

Framework

- ▶ requirements for abstract semantics
- ▶ abstract states requirements for abstract states
- ▶ specification of initial states
- ▶ default approximations

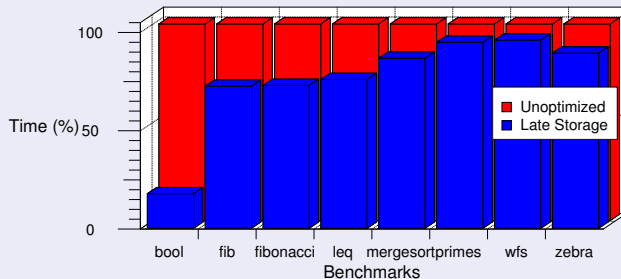
First Two Instantiations

- ▶ late storage analysis
avoids constraint store management overhead
- ▶ groundness analysis
well-known Prolog analysis ported to CHR

Analysis Results

24/34

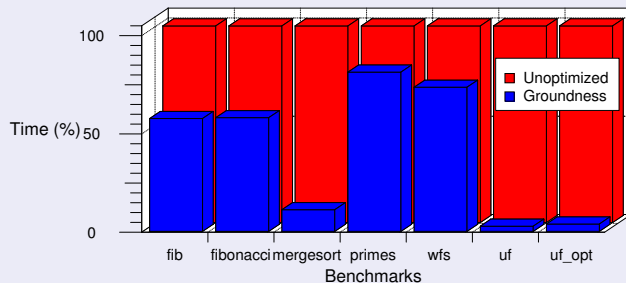
Late Storage



Groundness

Late Storage

Groundness



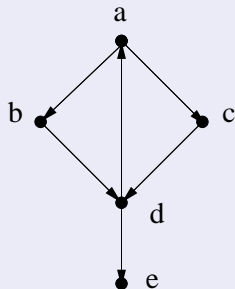
Contribution 3: CHR and Tabled Execution

25/34

Plain Prolog

```
reach(X,Y) :-
    edge(X,Y).
reach(X,Y) :-
    edge(X,Z),
    reach(Z,Y).
```

- ▶ redundant computation
- ▶ non-termination



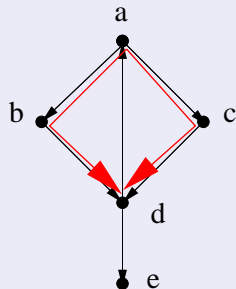
Contribution 3: CHR and Tabled Execution

25/34

Plain Prolog

```
reach(X,Y) :-
    edge(X,Y).
reach(X,Y) :-
    edge(X,Z),
    reach(Z,Y).
```

- ▶ redundant computation
- ▶ non-termination



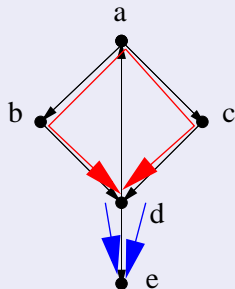
Contribution 3: CHR and Tabled Execution

25/34

Plain Prolog

```
reach(X,Y) :-
    edge(X,Y).
reach(X,Y) :-
    edge(X,Z),
    reach(Z,Y).
```

- ▶ redundant computation
- ▶ non-termination



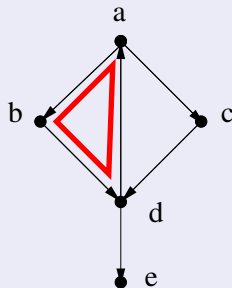
Contribution 3: CHR and Tabled Execution

25/34

Plain Prolog

```
reach(X,Y) :-
    edge(X,Y).
reach(X,Y) :-
    edge(X,Z),
    reach(Z,Y).
```

- ▶ redundant computation
- ▶ non-termination



XSB: Tabled Execution (D.S. Warren)

- ▶ tabulate answers
- ▶ compute fixed point
- ▶ avoids needless recomputation
- ▶ avoids infinite loops
- ▶ many applications: model checking, program analysis, parsing,
...

Problem

- ▶ no substantial support for constraints

CHR and Tabled Execution

27/34

Solution

- ▶ CHR for constraint solvers
- ▶ integrate CHR with tabled execution

General Integration of CHR and Tabling

- ▶ highlevel interface
- ▶ supports program-specific customizations
- ▶ handles all interactions between CHR constraint store and tables

Why Implication Checking?

- ▶ building block for complex constraints
 - ▶ conditional constraint combinators (Mozart)
 - ▶ reified constraints (`c1p(FD)`)
 $X = 1 \Rightarrow Y = 2$
- ▶ modular composition of constraint solvers
programming in the large

Implication Checking

29/34

- ▶ Logical Constraint Theory CT
 e.g. equality (=) with axioms

$$\begin{aligned} \forall x : x = x & \quad (\text{refl.}) \\ \forall x, y : x = y \rightarrow y = x & \quad (\text{symm.}) \\ \forall x, y, z : x = y \wedge y = z \rightarrow x = z & \quad (\text{trans.}) \end{aligned}$$

- ▶ Conjunction C of constraints that hold e.g.

$$C \equiv x = y \wedge y = z$$

- ▶ Query C for constraint of interest d :

$$CT \models C \rightarrow d$$

$$d \equiv z = x:$$

$$CT \models x = y \wedge y = z \rightarrow z = x$$

e.g.

Implication Checking for CHR

30/34

CHR Constraint Solver

- ▶ (partial) implementation of CT detects consistency (solutions)
- ▶ derives solved form from constraint store C
 $C \rightsquigarrow^* C'$ with $CT \models C \leftrightarrow C'$

Idea (with T. Frühwirth, G. Duck and P. Stuckey)

- ▶ exploit redundancy elimination in the solver:

$$\text{solve}(C) \equiv \text{solve}(C \wedge d) \Rightarrow C \rightarrow d$$

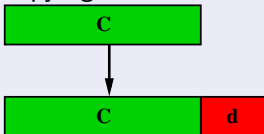
- ▶ sound for all solvers
- ▶ complete for canonical solvers
 and also for some non-canonical solvers

Automatic Implication Checking for CHR

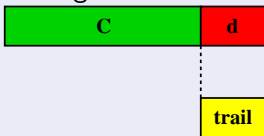
31/34

Two implementations compared

- ▶ copying



- ▶ trailing as a source-to-source transformation



- ▶ about the same performance
- ▶ allows building of solver hierarchies



- 1 Introduction
 - Constraint Handling Rules
 - Illustration: Show Cases of CHR
 - Context
 - Goals
- 2 Contributions
 - The K.U.Leuven CHR System
 - Abstract Interpretation
 - CHR and Tabled Execution
 - Automatic Implication Checking
- 3 Conclusion
 - Future Work on CHR

Major Contributions:

- ▶ a state-of-the-art CHR implementation. . .
- ▶ . . . with new optimizations
- ▶ a program analysis framework for CHR
- ▶ integration with tabled execution
- ▶ automatic implication checking for CHR solvers

2 Other Topics

- ▶ PARMA trailing
- ▶ Prolog refactoring

Usability Improvements

- ▶ verification analyses & tools
- ▶ reuse patterns
- ▶ constraint solver support
- ▶ more host languages

Efficiency Improvements

- ▶ scalability
- ▶ different execution strategies: parallel, user-defined, ...