

An Introduction to Constraint Handling Rules

Tom Schrijvers

tom.schrijvers@cs.kuleuven.ac.be

K.U.Leuven, Belgium



Overview

- 1. Constraint Solving, Alternatives
- 2. CHR Syntax and Semantics
- 3. Applications
- 4. Research
- 5. Conclusion

Constraint Solving Alternatives

Constraints in Prolog:

- Term equality constraint $=/2$

?- $X = f(Y)$.

?- $f(Y) = X$.

?- $X = f(Y), X = g(Y)$.

?- $X = f, Y = X$.

?- $X = Y, X = f$.

- Properties

- ◆ binding: remember constraint

- ◆ failure: inconsistent constraints

- ◆ order of constraints not important (!)

Constraint Solving Alternatives

Arithmetic in Prolog

- `is/2` is not a constraint

```
?- X = 1, Y = 2, Z is X + Y.
```

```
?- X = 1, Y = 2, Z = 4, Z is X + Y.
```

```
?- Z is X + Y, X = 1, Y = 2.
```

```
?- Z = 3, X = 1, Z is X + Y.
```

- Properties

- ◆ one-way

- ◆ order important

- ◆ sufficient information necessary

Constraint Solving Alternatives

How to add full arithmetic constraints to Prolog?

- Build them into the Prolog system
 - ◆ Interaction with $=/2$
 - ◆ System implementors only
 - ◆ No custom constraints / domains
 - adapt problem to available domains
e.g. use integers instead of atoms or booleans

Constraint Solving Alternatives

How to add user constraint domains to Prolog?

- Attributed Variables
 - ◆ keep constraints on variables
 - ◆ hook for $=/2$
 - ◆ low-level
 - very flexible
 - hard to write
 - hard to extend
 - hard to understand

Constraint Solving Alternatives

Constraint Handling Rules (Thom Frühwirth)

- high-level
- focus mainly on what, not how
- logic separated in different rules
- automatic scheduling
- easy to understand, extend and experiment with

CHR Syntax and Semantics

CHR consists of

- Constraints
 - ◆ terms with arguments, *e.g.* $leq(X,Y)$
 - ◆ can be called like $=/2$ or predicates
 - ◆ no set semantics
 - ◆ `:-` constraints $leq/2$, $geq/2$.
- Constraint Store: where the constraints are remembered (suspended)
- Rules
 - ◆ behavior and interaction of the constraints
 - ◆ 3 kinds of rules

CHR Syntax and Semantics

The Simplification Rule

- Syntax: $\text{head} \Leftrightarrow \text{guard} \mid \text{body} .$
 - ◆ *head*: conjunction of constraints
 - ◆ *guard*: Prolog goal, optional
 - ◆ *body*: any Prolog goal
- Examples

$\text{leq}(X, Y) , \text{leq}(Y, X) \Leftrightarrow X = Y .$

$\text{leq}(N, M) \Leftrightarrow \text{number}(N) , \text{number}(M) ,$
 $N =< M \mid \text{true} .$

CHR Syntax and Semantics

The Simplification Rule

- Syntax: $\text{head} \Leftrightarrow \text{guard} \mid \text{body} .$
- Semantics: constraints of the head are *replaced* with body
 - ◆ left to right only !
 - ◆ only matching in the head, no binding
 $\text{leq}(X, X) \Leftrightarrow \text{true} .$
 - ◆ no binding of head variables allowed in the guard

CHR Syntax and Semantics

The Propagation Rule

- Syntax: $\text{head} \implies \text{guard} \mid \text{body} .$
- Semantics: body is called once for head if guard succeeds
- Examples

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z) .$

CHR Syntax and Semantics

The Propagation Rule

- Syntax: $\text{head} \implies \text{guard} \mid \text{body} .$
- Difference with simplification:
 $\text{head} \Leftrightarrow \text{guard} \mid \text{head}, \text{body}.$
 - ◆ triggers infinitely many times
head
head, **head** , body
head, head, **head** , body, body
head, head, head, **head** , body, body, body
head, head, head, head, **head** , body, body,...

CHR Syntax and Semantics

The Simpagation Rule

- Syntax: $\text{head}_1 \setminus \text{head}_2 \Leftrightarrow \text{guard} \mid \text{body} .$
- Semantics: head_2 is replaced with body if guard succeeds
- Example

$$\begin{array}{l} \text{ub}(X, N) \setminus \text{ub}(X, M) \Leftrightarrow \\ N = < M \mid \text{true}. \end{array}$$

CHR Syntax and Semantics

The Simpagation Rule

- Syntax: $\text{head}_1 \setminus \text{head}_2 \Leftrightarrow \text{guard} \mid \text{body} .$
- Difference with simplification
 $\text{head}_1, \text{head}_2 \Leftrightarrow \text{guard} \mid \text{head}_1, \text{body}.$
 - ◆ Does not retrigger propagation rules.
 - ◆ Does not needlessly recheck other simplification/simpagation rules.

CHR Syntax and Semantics

A rule triggers 'when' it applies.

- Rules are tried when (active) constraint is called.
- Rules are tried and executed sequentially from top to bottom.
- Committed choice (no backtracking)
- Passive constraints are looked for among suspended constraints.

CHR Syntax and Semantics

A rule triggers 'when' it applies.

- Outcome is one of:
 - ◆ execution fails
 - ◆ active constraint is simplified away
 - ◆ active constraint suspends after last rule
- Suspended constraint triggers when argument is bound

Applications

- Simple less-than-or-equal constraint solver
:- constraints leq/2.

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: {}

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: {}

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: `{leq(X,Y)}`

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: `{leq(X,Y)}`

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: `{leq(X,Y)}`

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: `{leq(X,Y)}`

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: `{leq(X,Y), leq(Y,Z)}`

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: `{leq(X,Y), leq(Y,Z)}`

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(X, Z)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(X, Z)\}$

- Query:

?- $\text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: `{leq(X,Y), leq(Y,Z), leq(X,Z) }`

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \wedge \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(X, Z)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: `{leq(X,Y), leq(Y,Z), leq(X,Z) }`

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(X, Z)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(X, Z)\}$

- Query:

?- $\text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(X, Z)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(X, Z), \mathbf{\text{leq}(Z, X)}\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \mathbf{\text{leq}(Z, X)}.$

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: `{leq(X,Y), leq(Y,Z), leq(X,Z), leq(Z,X) }`

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \wedge \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(X, Z), \text{leq}(Z, X)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, Z), \text{leq}(Z, X).$

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \wedge \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, X)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, X), \text{leq}(X, X).$

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, X)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, X), \text{leq}(X, X).$

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: `{leq(X,Y), leq(Y,X)}`

- Query:

`?- leq(X,Y), leq(Y,X), leq(X,X).`

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \wedge \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: $\{\text{leq}(X, Y), \text{leq}(Y, X)\}$

- Query:

$?- \text{leq}(X, Y), \text{leq}(Y, X), \text{leq}(X, X).$

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: {}

- Query:

`?- leq(X,X), leq(X,X), leq(X,X).`

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: {}

- Query:

?- $\text{leq}(X, X), \text{leq}(X, X), \mathbf{\text{leq}(X, X)}.$

Applications

- Rules

$\text{leq}(X, X) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, X) \iff X = Y.$

$\text{leq}(X, Y) \setminus \text{leq}(X, Y) \iff \text{true}.$

$\text{leq}(X, Y), \text{leq}(Y, Z) \implies \text{leq}(X, Z).$

- Store: {}

- Query:

$?- \text{leq}(X, X), \text{leq}(X, X), \text{leq}(X, X).$

Applications

- Rules

`leq(X,X) <=> true.`

`leq(X,Y), leq(Y,X) <=> X = Y.`

`leq(X,Y) \ leq(X,Y) <=> true.`

`leq(X,Y), leq(Y,Z) ==> leq(X,Z).`

- Store: {}

- Query:

`?- leq(X,Y), leq(Y,Z), leq(Z,X).`

`X = Y = Z`

`Yes`

Applications

- Simple finite domain constraint solver

`:- constraints domain/2.`

`domain(X, []) => fail.`

`domain(X, [V]) => X = V.`

`domain(X, D1), domain(X, D2) =>
intersection(D1, D2, D),
domain(X, D).`

Applications

Non-constraint solving applications

- CHR can be used as a general purpose language
- Example

```
:- constraints fib/2.
```

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

```
fib(0,M) ==> M = 1.
```

```
fib(1,M) ==> 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.
```

Applications

Non-constraint solving applications

- Coroutine tasks

```
:- constraints freeze/2.
```

```
freeze(X,G) <=> nonvar(X) |  
                call(G).
```

```
freeze(X,G1), freeze(X,G2) <=>  
    freeze(X,(G1,G2)).
```

Applications

Some real world applications:

- solving clock values of timed automata
- well founded semantics of logic programs
- type analysis and type system extensions
- the Munich rent advisor (expert system)
- CHR compiler
- And more: computational linguistics, planning, scheduling, security policy analysis, ...

Research into CHR

- Compiled to Prolog (attributed variables, Christian Holzbaaur)
- CHR systems for: SICStus / Yap, ECLiPSe, Hal, Haskell, Java
- hProlog
- XSB: integrated with tabling
- Formal semantics, soundness, completeness
- Confluence, Termination
- Future work: optimized compilation (analysis)

Conclusion

CHR is

- compact
- rule-based
- constraint specification/implementation language
- easy to write, understand and modify
- something for you?