
Reasoning with, about and for Constraint Handling Rules

Slim Abdennadher
LMU München

Christophe Rigotti
INSA Lyon

22.7.2002

Constraint Programming

Example numeric lock

0 1 2 3 4 5 6 7 8 9

Greater or equal 5

Prime number

Constraint Programming

Example numeric lock

0 1 2 3 4 5 6 7 8 9

Greater or equal 5

Prime number

- **Declarative problem representation** by variables and constraints:

$$x \in \{0, 1, \dots, 9\} \wedge x \geq 5 \wedge \text{prime}(x)$$

Constraint Programming

Example numeric lock

0 1 2 3 4 5 6 7 8 9

Greater or equal 5

Prime number

- **Declarative problem representation** by variables and constraints:

$$x \in \{0, 1, \dots, 9\} \wedge x \geq 5 \wedge \text{prime}(x)$$

- **Constraint solving** reduce search space:

$$x \in \{0, 1, \dots, 9\} \wedge x \geq 5 \rightarrow x \in \{5, 6, 7, 8, 9\}$$

$$x \in \{5, 6, 7, 8, 9\} \wedge \text{prime}(x) \rightarrow x \in \{5, 7\}$$

Constraint Programming

Example numeric lock

0 1 2 3 4 5 6 7 8 9

Greater or equal 5

Prime number

- **Declarative problem representation** by variables and constraints:

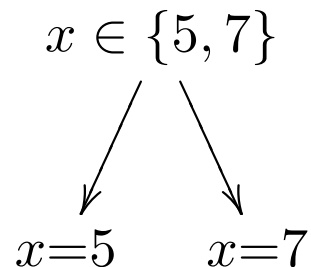
$$x \in \{0, 1, \dots, 9\} \wedge x \geq 5 \wedge \text{prime}(x)$$

- **Constraint solving** reduce search space:

$$x \in \{0, 1, \dots, 9\} \wedge x \geq 5 \rightarrow x \in \{5, 6, 7, 8, 9\}$$

$$x \in \{5, 6, 7, 8, 9\} \wedge \text{prime}(x) \rightarrow x \in \{5, 7\}$$

- **Search:**



Constraint Programming

Crypto-arithmetic Problem

$$\begin{array}{rcccccc} & & & S & E & N & D \\ & & & M & O & R & E \\ + & & & & & & \\ \hline = & M & O & N & E & Y \end{array}$$

Constraint Programming

Crypto-arithmetic Problem

$$\begin{array}{rcccccc} & & & S & E & N & D \\ & & & M & O & R & E \\ + & & & & & & \\ \hline = & M & O & N & E & Y & \end{array}$$

`solve(S,E,N,D,M,O,R,Y) :-`

Constraint Programming

Crypto-arithmetic Problem

$$\begin{array}{rcccccc} & & & S & E & N & D \\ & & & M & O & R & E \\ + & & & M & O & R & E \\ \hline = & M & O & N & E & Y & \end{array}$$

```
solve(S,E,N,D,M,O,R,Y) :-  
    [S,E,N,D,M,O,R,Y] in 0..9,
```

Constraint Programming

Crypto-arithmetic Problem

$$\begin{array}{rcccccc} & & & S & E & N & D \\ & & & M & O & R & E \\ + & & & & & & \\ \hline = & M & O & N & E & Y & \end{array}$$

```
solve(S,E,N,D,M,O,R,Y) :-  
    [S,E,N,D,M,O,R,Y] in 0..9,  
    S≠0, M ≠0,
```

Constraint Programming

Crypto-arithmetic Problem

$$\begin{array}{rcccccc} & & & S & E & N & D \\ & & & M & O & R & E \\ + & & & & & & \\ \hline = & M & O & N & E & Y & \end{array}$$

```
solve(S,E,N,D,M,O,R,Y) :-  
    [S,E,N,D,M,O,R,Y] in 0..9,  
    S≠0, M ≠0,  
    alldifferent([S,E,N,D,M,O,R,Y]),
```

Constraint Programming

Crypto-arithmetic Problem

$$\begin{array}{rcccccc} & & & S & E & N & D \\ & & & M & O & R & E \\ + & & & & & & \\ \hline = & M & O & N & E & Y \end{array}$$

`solve(S,E,N,D,M,O,R,Y) :-`

`[S,E,N,D,M,O,R,Y] in 0..9,`

`S≠0, M ≠0,`

`alldifferent([S,E,N,D,M,O,R,Y]),`

`1000*S + 100*E + 10*N + D`

`+ 1000*M + 100*O + 10*R + E`

`= 10000*M + 1000*O + 100*N + 10*E + Y`

Constraint Programming

Crypto-arithmetic Problem

$$\begin{array}{rcccccc} & & & S & E & N & D \\ + & & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

`solve(S,E,N,D,M,O,R,Y) :-`

`[S,E,N,D,M,O,R,Y] in 0..9,`

`S≠0, M ≠0,`

`alldifferent([S,E,N,D,M,O,R,Y]),`

`1000*S + 100*E + 10*N + D`

`+ 1000*M + 100*O + 10*R + E`

`= 10000*M + 1000*O + 100*N + 10*E + Y`

Constraint solving:

`S=9, M=1, O=0, E in 4..7, N in 5..8, [D,R,Y] in 2..8`

Constraint Programming

Crypto-arithmetic Problem

$$\begin{array}{rcccccc} & & & S & E & N & D \\ & & & M & O & R & E \\ + & & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

```
solve(S,E,N,D,M,O,R,Y) :-  
    [S,E,N,D,M,O,R,Y] in 0..9,  
    S≠0, M ≠0,  
    alldifferent([S,E,N,D,M,O,R,Y]),  
    1000*S + 100*E + 10*N + D  
+    1000*M + 100*O + 10*R + E  
= 10000*M + 1000*O + 100*N + 10*E + Y,  
    labeling([S,E,N,D,M,O,R,Y]).
```

Constraint solving:

S=9, M=1, O=0, E in 4..7, N in 5..8, [D,R,Y] in 2..8

Search:

S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2

Constraint Solving

Classical Approach: Constraint solver is a black-box

Constraint Solving

Classical Approach: Constraint solver is a black-box

Problems:

- hard to modify
- hard to build a solver over a new domain
- hard to reason about and analyze

Constraint Solving

Classical Approach: Constraint solver is a black-box

Problems:

- hard to modify
- hard to build a solver over a new domain
- hard to reason about and analyze

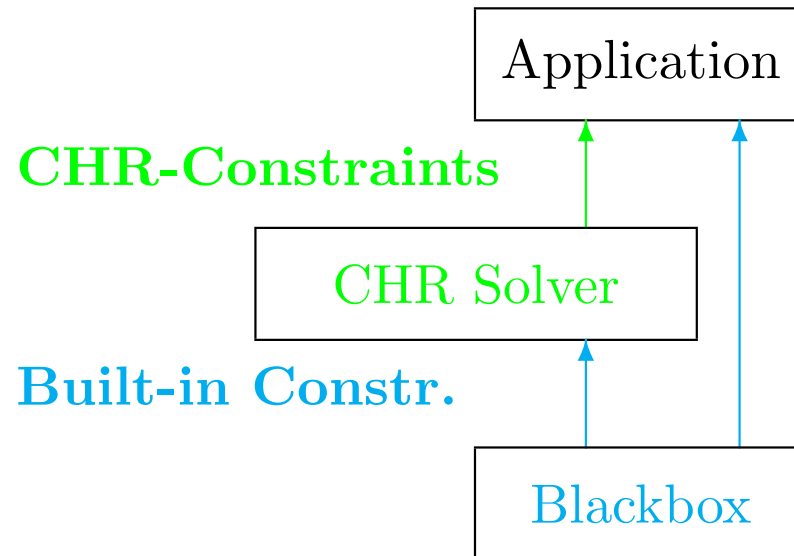
Proposal: Constraint Handling Rules (CHR)

Overview

- Introduction of CHR: Syntax and Semantics
- Reasoning about CHR: Program Analysis
- Reasoning with CHR: Implementation
- Reasoning for CHR: Automatic Generation of Constraint Solvers

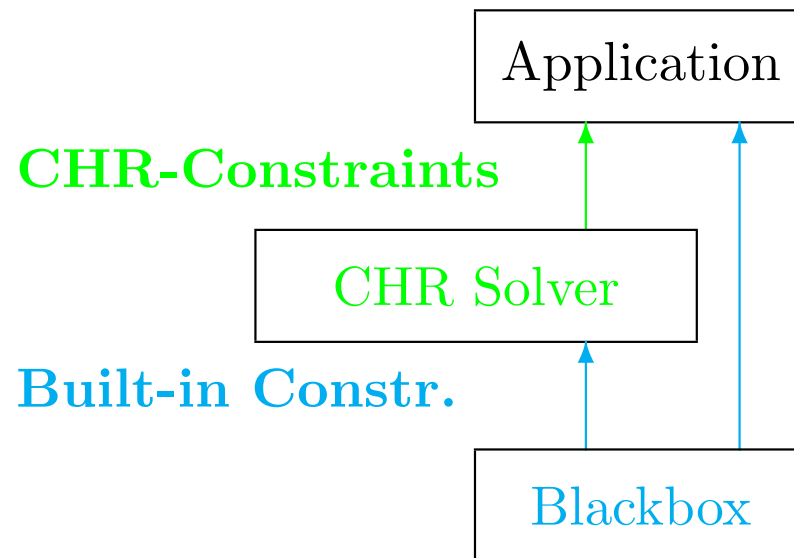
Constraint Handling Rules (CHR)

A concurrent constraint language with ask and tell (50+ applications)



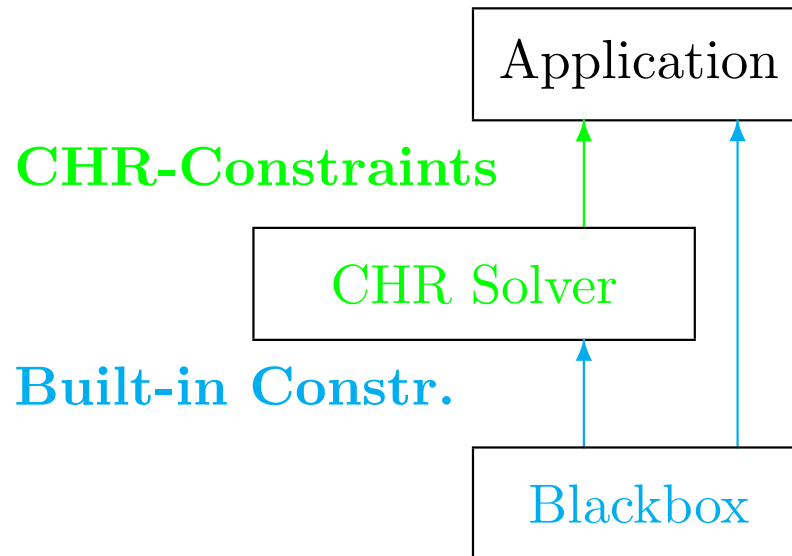
Constraint Handling Rules (CHR)

A concurrent constraint language with ask and tell (50+ applications)
for computational logic and more...



Constraint Handling Rules (CHR)

A concurrent constraint language with ask and tell (50+ applications)
for computational logic and more...



Extensions: Disjunction/Search, Dynamic Constraints, ...

CHR: Introductory Example \leq

$X \leq X \Leftrightarrow \text{true}$ (reflexivity)

$X \leq Y \wedge Y \leq X \Leftrightarrow X = Y$ (antisymmetry)

$X \leq Y \wedge Y \leq Z \Rightarrow X \leq Z$ (transitivity)

CHR: Introductory Example \leq

$$X \leq X \Leftrightarrow \text{true} \quad (\text{reflexivity})$$

$$X \leq Y \wedge Y \leq X \Leftrightarrow X = Y \quad (\text{antisymmetry})$$

$$X \leq Y \wedge Y \leq Z \Rightarrow X \leq Z \quad (\text{transitivity})$$

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A$$

CHR: Introductory Example \leq

$$X \leq X \Leftrightarrow \text{true} \quad (\text{reflexivity})$$

$$X \leq Y \wedge Y \leq X \Leftrightarrow X = Y \quad (\text{antisymmetry})$$

$$X \leq Y \wedge Y \leq Z \Rightarrow X \leq Z \quad (\text{transitivity})$$

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A$$

↓

(transitivity)

$$A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C}$$

CHR: Introductory Example \leq

$$\underline{X \leq X} \Leftrightarrow \text{true} \quad (\text{reflexivity})$$

$$\underline{X \leq Y} \wedge \underline{Y \leq X} \Leftrightarrow X = Y \quad (\text{antisymmetry})$$

$$\underline{X \leq Y} \wedge \underline{Y \leq Z} \Rightarrow \underline{X \leq Z} \quad (\text{transitivity})$$

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{C \leq A}$$

↓

(transitivity)

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{C \leq A} \wedge \underline{A \leq C}$$

↓

(antisymmetry)

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{A = C}$$

CHR: Introductory Example \leq

$$\underline{X \leq X} \Leftrightarrow \text{true} \quad (\text{reflexivity})$$

$$\underline{X \leq Y} \wedge \underline{Y \leq X} \Leftrightarrow \underline{X = Y} \quad (\text{antisymmetry})$$

$$\underline{X \leq Y} \wedge \underline{Y \leq Z} \Rightarrow \underline{X \leq Z} \quad (\text{transitivity})$$

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{C \leq A}$$

↓

(transitivity)

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{C \leq A} \wedge \underline{A \leq C}$$

↓

(antisymmetry)

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{A = C}$$

↓

(Black-box solver)

$$\underline{A \leq B} \wedge \underline{B \leq A} \wedge \underline{A = C}$$

CHR: Introductory Example \leq

$$X \leq X \Leftrightarrow \text{true} \quad (\text{reflexivity})$$

$$X \leq Y \wedge Y \leq X \Leftrightarrow X = Y \quad (\text{antisymmetry})$$

$$X \leq Y \wedge Y \leq Z \Rightarrow X \leq Z \quad (\text{transitivity})$$

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{C \leq A}$$

↓

(transitivity)

$$A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C}$$

↓

(antisymmetry)

$$A \leq B \wedge B \leq C \wedge \underline{A = C}$$

↓

(Black-box solver)

$$\underline{A \leq B} \wedge \underline{B \leq A} \wedge A = C$$

↓

(antisymmetry)

$$A = B \wedge A = C$$

CHR: Syntax and Declarative Semantics

Simplification rule: $H \Leftrightarrow C \mid B$

Propagation rule: $H \Rightarrow C \mid B$

- H : non-empty conjunction of user-defined constraints
- C : conjunction of built-in constraints
- B : conjunction of user-defined and built-in constraints

CHR: Syntax and Declarative Semantics

Simplification rule: $H \Leftrightarrow C \mid B$ $\forall \bar{x} (C \rightarrow (H \leftrightarrow \exists \bar{y} B))$

Propagation rule: $H \Rightarrow C \mid B$ $\forall \bar{x} (C \rightarrow (H \rightarrow \exists \bar{y} B))$

- H : non-empty conjunction of user-defined constraints
- C : conjunction of built-in constraints
- B : conjunction of user-defined and built-in constraints

Declarative semantics of a CHR program:

- the above logical formulas
- a constraint theory CT for the built-in constraints.

Simplify

If $(H \Leftrightarrow C \mid B)$ is a fresh variant of a rule with variables \bar{x}

and $CT \models G_{built} \rightarrow \exists \bar{x}(H=H' \wedge C)$

then $H' \wedge G \mapsto G \wedge H=H' \wedge B$

CHR: Operational Semantics

Simplify

If $(H \Leftrightarrow C \mid B)$ is a fresh variant of a rule with variables \bar{x}

and $CT \models G_{built} \rightarrow \exists \bar{x}(H=H' \wedge C)$

then $H' \wedge G \mapsto G \wedge H=H' \wedge B$

Propagate

If $(H \Rightarrow C \mid B)$ is a fresh variant of a rule with variables \bar{x}

and $CT \models G_{built} \rightarrow \exists \bar{x}(H=H' \wedge C)$

then $H' \wedge G \mapsto H' \wedge G \wedge H=H' \wedge B$

Anytime Algorithm

Computation can be interrupted and restarted.

Intermediate results approximate final result.

Anytime Algorithm

Computation can be interrupted and restarted.
Intermediate results approximate final result.

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{C \leq A}$$

Anytime Algorithm

Computation can be interrupted and restarted.
Intermediate results approximate final result.

$$\begin{array}{c} \underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A \\ \downarrow \\ A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C} \end{array} \quad \text{(transitivity)}$$

Anytime Algorithm

Computation can be interrupted and restarted.
Intermediate results approximate final result.

$$\begin{array}{l} \underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A \\ \downarrow \text{ (transitivity) } \\ A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C} \\ \downarrow \text{ (antisymmetry) } \\ A \leq B \wedge B \leq C \wedge \underline{A = C} \end{array}$$

Anytime Algorithm

Computation can be interrupted and restarted.

Intermediate results approximate final result.

$$\begin{array}{l} \underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A \\ \downarrow \text{ (transitivity) } \\ A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C} \\ \downarrow \text{ (antisymmetry) } \\ A \leq B \wedge B \leq C \wedge \underline{A = C} \\ \downarrow \text{ (Black-box solver) } \\ \underline{A \leq B} \wedge \underline{B \leq A} \wedge A = C \end{array}$$

Anytime Algorithm

Computation can be interrupted and restarted.
Intermediate results approximate final result.

$$\begin{array}{l} \underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A \\ \downarrow \text{ (transitivity) } \\ A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C} \\ \downarrow \text{ (antisymmetry) } \\ A \leq B \wedge B \leq C \wedge \underline{A = C} \\ \downarrow \text{ (Black-box solver) } \\ \underline{A \leq B} \wedge \underline{B \leq A} \wedge A = C \\ \downarrow \text{ (antisymmetry) } \\ A = B \wedge A = C \end{array}$$

Monotonicity and Incrementality

If $G \longmapsto G'$
then $G \wedge C \longmapsto G' \wedge C$

Monotonicity and Incrementality

$$\begin{array}{l} \text{If } G \longmapsto G' \\ \text{then } G \wedge C \longmapsto G' \wedge C \end{array}$$

Online Algorithm

The complete input is initially unknown.

The input data arrives during computation.

No recomputation from scratch necessary.

Monotonicity and Incrementality

$$\begin{array}{l} \text{If } G \longmapsto G' \\ \text{then } G \wedge C \longmapsto G' \wedge C \end{array}$$

Online Algorithm

The complete input is initially unknown.

The input data arrives during computation.

No recomputation from scratch necessary.

$$\underline{A \leq B} \wedge \underline{B \leq C}$$

Monotonicity and Incrementality

$$\begin{array}{l} \text{If } G \longmapsto G' \\ \text{then } G \wedge C \longmapsto G' \wedge C \end{array}$$

Online Algorithm

The complete input is initially unknown.

The input data arrives during computation.

No recomputation from scratch necessary.

$$\begin{array}{c} \underline{A \leq B} \wedge \underline{B \leq C} \\ \downarrow \\ \underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{A \leq C} \end{array} \quad \text{(transitivity)}$$

Monotonicity and Incrementality

$$\begin{array}{l} \text{If } G \longmapsto G' \\ \text{then } G \wedge C \longmapsto G' \wedge C \end{array}$$

Online Algorithm

The complete input is initially unknown.

The input data arrives during computation.

No recomputation from scratch necessary.

$$\begin{array}{c} \underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A \\ \downarrow \\ A \leq B \wedge B \leq C \wedge \underline{A \leq C} \wedge \underline{C \leq A} \end{array} \quad \text{(transitivity)}$$

Monotonicity and Incrementality

$$\begin{array}{l} \text{If } G \longmapsto G' \\ \text{then } G \wedge C \longmapsto G' \wedge C \end{array}$$

Online Algorithm

The complete input is initially unknown.

The input data arrives during computation.

No recomputation from scratch necessary.

$$\begin{array}{l} \underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A \\ \downarrow \text{ (transitivity) } \\ A \leq B \wedge B \leq C \wedge \underline{A \leq C} \wedge \underline{C \leq A} \\ \downarrow \text{ (antisymmetry) } \\ A \leq B \wedge B \leq C \wedge \underline{A = C} \\ \downarrow \\ \dots \end{array}$$

Concurrency

If $A \longmapsto B$
and $C \longmapsto D$
then $A \wedge C \longmapsto B \wedge D$

Concurrency

If $A \longmapsto B$
and $C \longmapsto D$
then $A \wedge C \longmapsto B \wedge D$

$$\underline{A \leq B} \wedge \underline{B \leq C} \quad \wedge \quad C \leq D \wedge D \leq A$$

Concurrency

If $A \longmapsto B$
and $C \longmapsto D$
then $A \wedge C \longmapsto B \wedge D$

(transitivity) $\frac{\underline{A \leq B} \wedge \underline{B \leq C}}{\downarrow} \wedge \frac{C \leq D \wedge D \leq A}{\downarrow}$ (transitivity)
 $\underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{A \leq C} \wedge C \leq D \wedge D \leq A \wedge \underline{C \leq A}$

Concurrency

If $A \longmapsto B$
and $C \longmapsto D$
then $A \wedge C \longmapsto B \wedge D$

(transitivity) $\frac{\underline{A \leq B} \wedge \underline{B \leq C}}{\downarrow} \wedge \frac{C \leq D \wedge D \leq A}{\downarrow}$ (transitivity)

$\underline{A \leq B} \wedge \underline{B \leq C} \wedge \underline{A \leq C} \wedge C \leq D \wedge D \leq A \wedge \underline{C \leq A}$

\downarrow

...

Overview

- Introduction of CHR: Syntax and Semantics
- Reasoning about CHR: Program Analysis
- Reasoning with CHR: Implementation
- Reasoning for CHR: Automatic Generation of Constraint Solvers

CHR Program Analysis

Termination

Every computation starting from any goal ends. [LNAI 1865]

Consistency

The logical meaning of the rules is consistent. [Constraints Journal 2000]

Confluence

The answer of a query is always the same, no matter which of the applicable rules are applied. [CP'96, CP'97, Constraints Journal 2000]

Completion

Make non-confluent programs confluent by adding new rules. [CP'98]

Operational Equivalence

Do two programs have the same behavior? [CP'99]

Complexity

Determine time complexity from structure of rules. [KR'02]

Minimal States

For each rule, there is a minimal, most general state to which it is applicable.

Minimal States

For each rule, there is a minimal, most general state to which it is applicable.

Rule: $H \Leftrightarrow C \mid B$ or $H \Rightarrow C \mid B$

Minimal State: $H \wedge C$

Minimal States

For each rule, there is a minimal, most general state to which it is applicable.

Rule: $H \Leftrightarrow C \mid B$ or $H \Rightarrow C \mid B$

Minimal State: $H \wedge C$

Every other state to which the rule is applicable contains the minimal state (cf. Monotonicity/Incrementality).

Confluence

Given a goal, every computation leads to the same result no matter what rules are applied.

Confluence

Given a goal, every computation leads to the same result no matter what rules are applied.

A decidable, sufficient and necessary condition for confluence of terminating CHR programs through joinability of critical pairs.

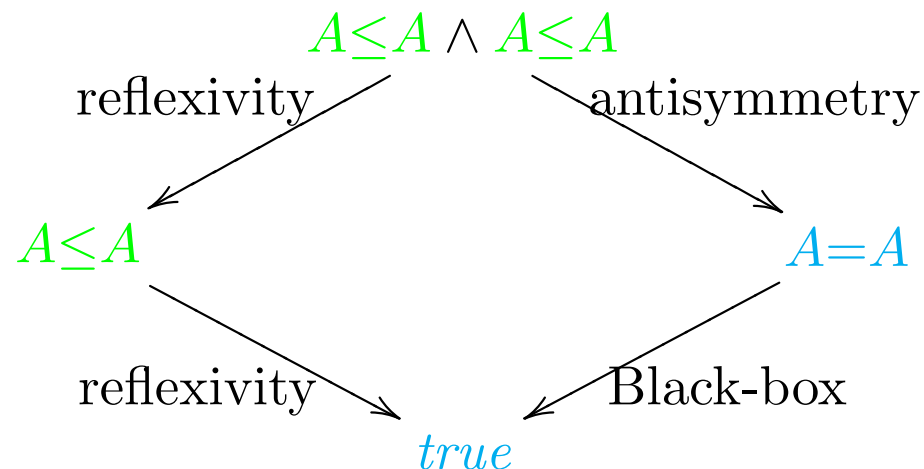
Confluence

Given a goal, every computation leads to the same result no matter what rules are applied.

A decidable, sufficient and necessary condition for confluence of terminating CHR programs through joinability of critical pairs.

$$\begin{aligned} X \leq X &\Leftrightarrow \text{true} && \text{(reflexivity)} \\ X \leq Y \wedge Y \leq X &\Leftrightarrow X = Y && \text{(antisymmetry)} \end{aligned}$$

Start from two overlapping minimal states



Completion

Derive rules from a non-joinable critical pair for transition from one of the critical states into the other one.

Completion

Derive rules from a non-joinable critical pair for transition from one of the critical states into the other one.

$$X \leq Y \wedge Y \leq X \Leftrightarrow X = Y \quad (\text{antisymmetry})$$

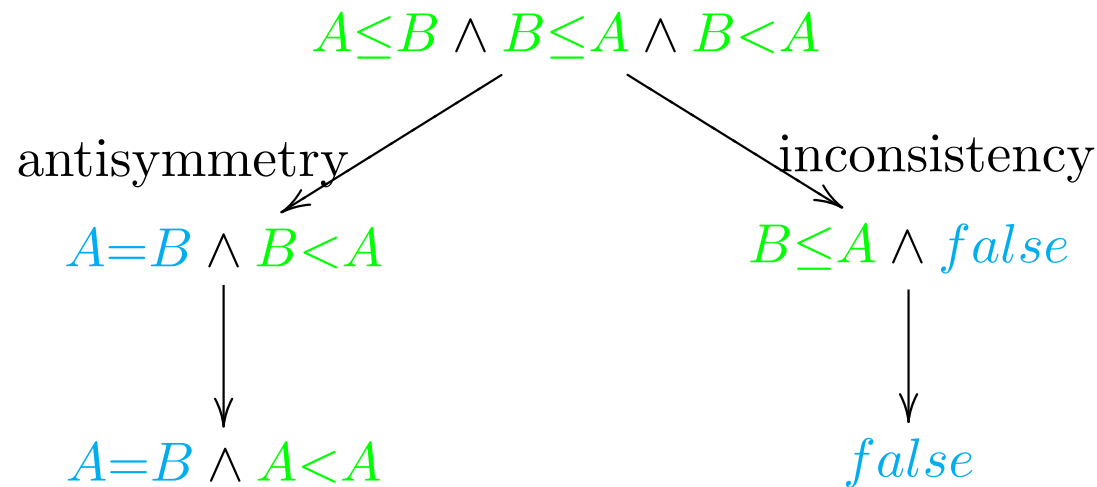
$$X \leq Y \wedge Y < X \Leftrightarrow \textit{false} \quad (\text{inconsistency})$$

Completion

Derive rules from a non-joinable critical pair for transition from one of the critical states into the other one.

$$X \leq Y \wedge Y \leq X \Leftrightarrow X = Y \quad (\text{antisymmetry})$$

$$X \leq Y \wedge Y < X \Leftrightarrow \text{false} \quad (\text{inconsistency})$$

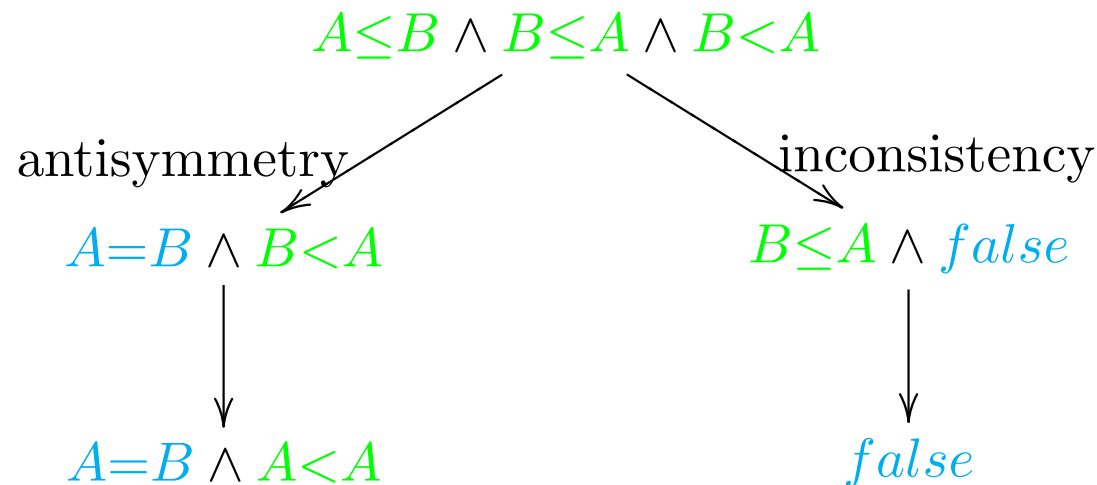


Completion

Derive rules from a non-joinable critical pair for transition from one of the critical states into the other one.

$$X \leq Y \wedge Y \leq X \Leftrightarrow X = Y \quad (\text{antisymmetry})$$

$$X \leq Y \wedge Y < X \Leftrightarrow \text{false} \quad (\text{inconsistency})$$



$$X < X \Leftrightarrow \text{false} \quad (\text{irreflexivity})$$

Operational Equivalence

Given a goal and two programs, the results of the computation in both programs are the same.

Operational Equivalence

Given a goal and two programs, the results of the computation in both programs are the same.

A decidable, sufficient and necessary condition for operational equivalence of terminating CHR programs through joinability of minimal states.

Operational Equivalence

Given a goal and two programs, the results of the computation in both programs are the same.

A decidable, sufficient and necessary condition for operational equivalence of terminating CHR programs through joinability of minimal states.

P_1 defines max

$$max(X, Y, Z) \Leftrightarrow X < Y \mid Z = Y.$$

$$max(X, Y, Z) \Leftrightarrow X \geq Y \mid Z = X.$$

P_2 defines max

$$max(X, Y, Z) \Leftrightarrow X \leq Y \mid Z = Y.$$

$$max(X, Y, Z) \Leftrightarrow X > Y \mid Z = X.$$

$$max(X, Y, Z) \wedge X \geq Y$$

$\downarrow P_1$

$$Z = X \wedge X \geq Y$$

$$max(X, Y, Z) \wedge X \geq Y$$

$\downarrow P_2$

Overview

- Introduction of CHR: Syntax and Semantics
- Reasoning about CHR: Program Analysis
- Reasoning with CHR: Implementation
- Reasoning for CHR: Automatic Generation of Constraint Solvers

Boolean Constraints

- *Truth values:* 1 and 0
- *Connectives:* $\neg, \sqcap, \sqcup, \oplus, \rightarrow, \leftrightarrow$
- *Theory:*

X	Y	$\neg X$	$X \sqcap Y$	$X \sqcup Y$	$X \oplus Y$	$X \rightarrow Y$	$X \leftrightarrow Y$
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	0	0
1	1	0	1	1	0	1	1

- *Local consistency algorithm:* simplifies one atomic Boolean constraint at a time into one or more syntactic equalities whenever possible.

Boolean Constraints: Solver

$$\text{and}(X, X, Z) \Leftrightarrow X=Z.$$

$$\text{and}(X, Y, 1) \Leftrightarrow X=1 \wedge Y=1.$$

$$\text{and}(X, 1, Z) \Leftrightarrow X=Z.$$

$$\text{and}(X, 0, Z) \Leftrightarrow Z=0.$$

$$\text{and}(1, Y, Z) \Leftrightarrow Y=Z.$$

$$\text{and}(0, Y, Z) \Leftrightarrow Z=0.$$

Boolean Constraints: Solver

$$\text{and}(X, X, Z) \Leftrightarrow X=Z.$$

$$\text{and}(X, Y, 1) \Leftrightarrow X=1 \wedge Y=1.$$

$$\text{and}(X, 1, Z) \Leftrightarrow X=Z.$$

$$\text{and}(X, 0, Z) \Leftrightarrow Z=0.$$

$$\text{and}(1, Y, Z) \Leftrightarrow Y=Z.$$

$$\text{and}(0, Y, Z) \Leftrightarrow Z=0.$$

$$\text{neg}(X, X) \Leftrightarrow \text{false}.$$

$$\text{neg}(X, 0) \Leftrightarrow X=1.$$

$$\text{neg}(X, 1) \Leftrightarrow X=0.$$

$$\text{neg}(0, Y) \Leftrightarrow Y=1.$$

$$\text{neg}(1, Y) \Leftrightarrow Y=0.$$

Boolean Constraints: Solver

$$\text{and}(X, X, Z) \Leftrightarrow X=Z.$$

$$\text{and}(X, Y, 1) \Leftrightarrow X=1 \wedge Y=1.$$

$$\text{and}(X, 1, Z) \Leftrightarrow X=Z.$$

$$\text{and}(X, 0, Z) \Leftrightarrow Z=0.$$

$$\text{and}(1, Y, Z) \Leftrightarrow Y=Z.$$

$$\text{and}(0, Y, Z) \Leftrightarrow Z=0.$$

$$\text{neg}(X, X) \Leftrightarrow \text{false}.$$

$$\text{neg}(X, 0) \Leftrightarrow X=1.$$

$$\text{neg}(X, 1) \Leftrightarrow X=0.$$

$$\text{neg}(0, Y) \Leftrightarrow Y=1.$$

$$\text{neg}(1, Y) \Leftrightarrow Y=0.$$

$$\text{and}(X, Y, Z) \wedge \text{neg}(X, Y) \Leftrightarrow \text{neg}(X, Y) \wedge Z=0.$$

$$\text{and}(X, Y, Z) \wedge \text{neg}(X, Z) \Leftrightarrow X=1 \wedge Y=0 \wedge Z=0.$$

...

Syntactic Unification

- *Rational tree*: (possibly infinite) tree with finite set of subtrees, e.g. $X=f(X)$.
- *Solved normal form*:
 - *false* or
 - $X_1=t_1 \wedge \dots \wedge X_n=t_n$ ($n \geq 0$) where X_i is different to X_j and t_j , if $i \leq j$
- *Theory*:
 - Reflexivity*: $\forall(\text{true} \rightarrow x=x)$
 - Symmetry*: $\forall(x=y \rightarrow y=x)$
 - Transitivity*: $\forall(x=y \wedge y=z \rightarrow x=z)$
 - Compatibility*: $\forall(x_1=y_1 \wedge \dots \wedge x_n=y_n \rightarrow f(x_1, \dots, x_n)=f(y_1, \dots, y_n))$
 - Decomposition*: $\forall(f(x_1, \dots, x_n)=f(y_1, \dots, y_n) \rightarrow x_1=y_1 \wedge \dots \wedge x_n=y_n)$
 - Contradiction*: $\forall(f(x_1, \dots, x_n)=g(y_1, \dots, y_m) \rightarrow \text{false})$ if $f \neq g$ or $n \neq m$

Syntactic Unification: Solver

reflexivity	@	$X=X \Leftrightarrow \text{var}(X) \mid \text{true}.$
orientation	@	$T=X \Leftrightarrow \text{var}(X) \wedge X@<T \mid X=T.$
decomposition	@	$T1=T2 \Leftrightarrow \text{nonvar}(T1) \wedge \text{nonvar}(T2) \mid$ $\text{same_functor}(T1,T2) \wedge$ $\text{same_args}(T1,T2).$
confrontation	@	$X=T1 \wedge X=T2 \Leftrightarrow \text{var}(X) \wedge X@<T1 \wedge T1@=<T2 \mid$ $X=T1 \wedge T1=T2.$

$$\begin{array}{c} \text{h}(Y, f(a), g(X, a)) = \text{h}(f(U), Y, g(\text{h}(Y), U)) \\ \hline \text{h} \rightarrow \text{decomposition} \text{h} \rightarrow^* \\ Y = f(U) \wedge \underline{f(a) = Y} \wedge g(X, a) = g(\text{h}(Y), U) \end{array}$$

Syntactic Unification: Solver

reflexivity	@	$X=X \Leftrightarrow \text{var}(X) \mid \text{true}.$
orientation	@	$T=X \Leftrightarrow \text{var}(X) \wedge X@<T \mid X=T.$
decomposition	@	$T1=T2 \Leftrightarrow \text{nonvar}(T1) \wedge \text{nonvar}(T2) \mid$ $\text{same_functor}(T1,T2) \wedge$ $\text{same_args}(T1,T2).$
confrontation	@	$X=T1 \wedge X=T2 \Leftrightarrow \text{var}(X) \wedge X@<T1 \wedge T1@=<T2 \mid$ $X=T1 \wedge T1=T2.$

	$\frac{h(Y, f(a), g(X, a)) = h(f(U), Y, g(h(Y), U))}{Y = f(U) \wedge \underline{f(a) = Y} \wedge g(X, a) = g(h(Y), U)}$
$\mapsto_{\text{decomposition}} \mapsto^*$	$Y = f(U) \wedge Y = f(a) \wedge \underline{g(X, a) = g(h(Y), U)}$
$\mapsto_{\text{orientation}}$	
$\mapsto \dots$	

Syntactic Unification: Solver

reflexivity	@	$X=X \Leftrightarrow \text{var}(X) \mid \text{true}.$
orientation	@	$T=X \Leftrightarrow \text{var}(X) \wedge X@<T \mid X=T.$
decomposition	@	$T1=T2 \Leftrightarrow \text{nonvar}(T1) \wedge \text{nonvar}(T2) \mid$ $\text{same_functor}(T1,T2) \wedge$ $\text{same_args}(T1,T2).$
confrontation	@	$X=T1 \wedge X=T2 \Leftrightarrow \text{var}(X) \wedge X@<T1 \wedge T1@=<T2 \mid$ $X=T1 \wedge T1=T2.$

	$\frac{h(Y, f(a), g(X, a)) = h(f(U), Y, g(h(Y), U))}{Y=f(U) \wedge \underline{f(a)=Y} \wedge g(X, a) = g(h(Y), U)}$
$\mapsto_{\text{decomposition}} \mapsto^*$	$Y=f(U) \wedge Y=f(a) \wedge \underline{g(X, a) = g(h(Y), U)}$
$\mapsto_{\text{orientation}}$	
$\mapsto \dots$	
\mapsto	$Y=f(U) \wedge \underline{U=a} \wedge X=h(Y) \wedge \underline{U=a}$

Syntactic Unification: Solver

reflexivity	@	$X=X \Leftrightarrow \text{var}(X) \mid \text{true}.$
orientation	@	$T=X \Leftrightarrow \text{var}(X) \wedge X@<T \mid X=T.$
decomposition	@	$T1=T2 \Leftrightarrow \text{nonvar}(T1) \wedge \text{nonvar}(T2) \mid$ $\text{same_functor}(T1,T2) \wedge$ $\text{same_args}(T1,T2).$
confrontation	@	$X=T1 \wedge X=T2 \Leftrightarrow \text{var}(X) \wedge X@<T1 \wedge T1@=<T2 \mid$ $X=T1 \wedge T1=T2.$

		$\frac{h(Y, f(a), g(X, a)) = h(f(U), Y, g(h(Y), U))}{Y=f(U) \wedge \underline{f(a)=Y} \wedge g(X, a) = g(h(Y), U)}$
$\mapsto_{\text{decomposition}} \mapsto^*$		$Y=f(U) \wedge Y=f(a) \wedge \underline{g(X, a) = g(h(Y), U)}$
$\mapsto_{\text{orientation}}$		$Y=f(U) \wedge \underline{U=a} \wedge X=h(Y) \wedge \underline{U=a}$
$\mapsto \dots$		$Y=f(U) \wedge U=a \wedge X=h(Y) \wedge \underline{a=a}$
$\mapsto_{\text{confrontation}}$		

Syntactic Unification: Solver

reflexivity	@	$X=X \Leftrightarrow \text{var}(X) \mid \text{true}.$
orientation	@	$T=X \Leftrightarrow \text{var}(X) \wedge X@<T \mid X=T.$
decomposition	@	$T1=T2 \Leftrightarrow \text{nonvar}(T1) \wedge \text{nonvar}(T2) \mid$ $\text{same_functor}(T1,T2) \wedge$ $\text{same_args}(T1,T2).$
confrontation	@	$X=T1 \wedge X=T2 \Leftrightarrow \text{var}(X) \wedge X@<T1 \wedge T1@=<T2 \mid$ $X=T1 \wedge T1=T2.$

		$\frac{h(Y, f(a), g(X, a)) = h(f(U), Y, g(h(Y), U))}{Y=f(U) \wedge \underline{f(a)=Y} \wedge g(X, a) = g(h(Y), U)}$
$\mapsto_{\text{decomposition}} \mapsto^*$		$Y=f(U) \wedge Y=f(a) \wedge \underline{g(X, a) = g(h(Y), U)}$
$\mapsto_{\text{orientation}}$		$Y=f(U) \wedge \underline{U=a} \wedge X=h(Y) \wedge \underline{U=a}$
$\mapsto \dots$		$Y=f(U) \wedge U=a \wedge X=h(Y) \wedge \underline{a=a}$
$\mapsto_{\text{confrontation}}$		$Y=f(U) \wedge U=a \wedge X=h(Y)$
$\mapsto_{\text{decomposition}} \mapsto^*$		

Linear Polynomial Equations

- *Equations* of the form $a_1X_1 + \dots + a_nX_n + b = 0$.
- *Solved form*: Leftmost variable occurs only once.
- *Theory*: The linear existential fragment of Tarski's axiomatic theory of real closed fields for elementary geometry.
- Reach solved normal form by *variable elimination*.
 - Choose an equation $a_1 * X_1 + \dots + a_n * X_n + b = 0$
 - Make its left-most variable explicit:
$$X_1 = -(a_2 * X_2 + \dots + a_n * X_n + b)/a_1$$
 - Replace all other occurrences of X_1 by $-(a_2 * X_2 + \dots + a_n * X_n + b)/a_1$.
 - Simplify the resulting equations into allowed constraints (this is always possible).
 - Repeat until solved.

Linear Polynomial Equations: Solver

$A1*X+P1=0 \wedge XP=0 \Leftrightarrow$
`find(A2*X,XP,P2) |`
`canon(P2-(P1/A1)*A2,P3) ^`
 $A1*X+P1=0 \wedge P3=0.$

$B=0 \Leftrightarrow$ `number(B) | zero(B).`

Linear Polynomial Equations: Solver

```
A1*X+P1=0 ∧ XP=0 ⇔  
  find(A2*X,XP,P2) |  
  canon(P2-(P1/A1)*A2,P3) ∧  
  A1*X+P1=0 ∧ P3=0.
```

```
B=0 ⇔ number(B) | zero(B).
```

$$1*X+3*Y+5=0 \wedge 3*X+2*Y+8=0$$

Linear Polynomial Equations: Solver

```
A1*X+P1=0 ^ XP=0 ⇔  
  find(A2*X,XP,P2) |  
  canon(P2-(P1/A1)*A2,P3) ^  
  A1*X+P1=0 ^ P3=0.
```

```
B=0 ⇔ number(B) | zero(B).
```

```
1*X+3*Y+5=0 ^ 3*X+2*Y+8=0
```

```
canon((2*Y+8) - ((3*Y+5)/1)*3,P3) % P3=-7*Y+ -7
```

```
1*X+3*Y+5=0 ^ -7*Y+ -7=0 % Y=-1
```

Linear Polynomial Equations: Solver

```
A1*X+P1=0 ∧ XP=0 ⇔  
  find(A2*X,XP,P2) |  
  canon(P2-(P1/A1)*A2,P3) ∧  
  A1*X+P1=0 ∧ P3=0.
```

```
B=0 ⇔ number(B) | zero(B).
```

```
1*X+3*Y+5=0 ∧ 3*X+2*Y+8=0
```

```
canon((2*Y+8) - ((3*Y+5)/1)*3,P3) % P3=-7*Y+ -7
```

```
1*X+3*Y+5=0 ∧ -7*Y+ -7=0 % Y=-1
```

```
canon((1*X+5) - ((-7)/-7)*3,P3') % P3'=1*X+2
```

```
1*X+2=0 ∧ -7*Y+ -7=0 % X=-2
```

Propositional Resolution

- *Boolean CSP in CNF*: Conjunction of clauses
- *Clause*: Disjunction of Literals
- *Literal*: Positive or negative atomic proposition
- *Clause as ordered list of signed variables*.
E.g., $\neg x \vee y \vee z$ as $\text{cl}([-x, +y, +z])$.

CHR and Automated Reasoning

Propositional Resolution

- *Boolean CSP in CNF*: Conjunction of clauses
- *Clause*: Disjunction of Literals
- *Literal*: Positive or negative atomic proposition
- *Clause as ordered list of signed variables.*

E.g., $\neg x \vee y \vee z$ as `cl([-x,+y,+z])`.

`empty_clause @ cl([]) ⇔ false.`

`tautology @ cl(L) ⇔ in(+X,L) ∧ in(-X,L) | true.`

`resolution @ cl(L1) ∧ cl(L2) ⇒
 find(+X,L1,L3) ∧ find(-X,L2,L4) |
 merge(L3,L4,L) ∧
 cl(L).`

Bottom-up evaluation of logic programs

$$p(X, Y) \leftarrow e(X, Y).$$

$$p(X, Y) \leftarrow e(X, Z) \wedge p(Z, Y).$$

Bottom-up evaluation of logic programs

$$p(X, Y) \leftarrow e(X, Y).$$

$$p(X, Y) \leftarrow e(X, Z) \wedge p(Z, Y).$$

is transformed into

$$e(X, Y) \Rightarrow p(X, Y).$$

$$e(X, Z) \wedge p(Z, Y) \Rightarrow p(X, Y).$$

Bottom-up evaluation of logic programs

$$p(X, Y) \leftarrow e(X, Y).$$

$$p(X, Y) \leftarrow e(X, Z) \wedge p(Z, Y).$$

is transformed into

$$e(X, Y) \Rightarrow p(X, Y).$$

$$e(X, Z) \wedge p(Z, Y) \Rightarrow p(X, Y).$$

$$e(a, b) \wedge e(b, c) \wedge e(c, d)$$

Bottom-up evaluation of logic programs

$$p(X, Y) \leftarrow e(X, Y).$$

$$p(X, Y) \leftarrow e(X, Z) \wedge p(Z, Y).$$

is transformed into

$$e(X, Y) \Rightarrow p(X, Y).$$

$$e(X, Z) \wedge p(Z, Y) \Rightarrow p(X, Y).$$

$$e(a, b) \wedge e(b, c) \wedge e(c, d)$$

↓

$$e(a, b) \wedge e(b, c) \wedge e(c, d) \wedge p(a, b) \wedge p(b, c) \wedge p(c, d)$$

Bottom-up evaluation of logic programs

$$p(X, Y) \leftarrow e(X, Y).$$

$$p(X, Y) \leftarrow e(X, Z) \wedge p(Z, Y).$$

is transformed into

$$e(X, Y) \Rightarrow p(X, Y).$$

$$e(X, Z) \wedge p(Z, Y) \Rightarrow p(X, Y).$$

$$e(a, b) \wedge e(b, c) \wedge e(c, d)$$

↓

$$e(a, b) \wedge e(b, c) \wedge e(c, d) \wedge p(a, b) \wedge p(b, c) \wedge p(c, d)$$

↓

$$e(a, b) \wedge e(b, c) \wedge e(c, d) \wedge p(a, b) \wedge p(b, c) \wedge p(c, d) \wedge p(a, c) \wedge p(b, d)$$

Bottom-up evaluation of logic programs

$$p(X, Y) \leftarrow e(X, Y).$$

$$p(X, Y) \leftarrow e(X, Z) \wedge p(Z, Y).$$

is transformed into

$$e(X, Y) \Rightarrow p(X, Y).$$

$$e(X, Z) \wedge p(Z, Y) \Rightarrow p(X, Y).$$

$$e(a, b) \wedge e(b, c) \wedge e(c, d)$$

↓

$$e(a, b) \wedge e(b, c) \wedge e(c, d) \wedge p(a, b) \wedge p(b, c) \wedge p(c, d)$$

↓

$$e(a, b) \wedge e(b, c) \wedge e(c, d) \wedge p(a, b) \wedge p(b, c) \wedge p(c, d) \wedge p(a, c) \wedge p(b, d)$$

↓

$$e(a, b) \wedge e(b, c) \wedge e(c, d) \wedge p(a, b) \wedge p(b, c) \wedge p(c, d) \wedge p(a, c) \wedge p(b, d) \wedge p(a, d)$$

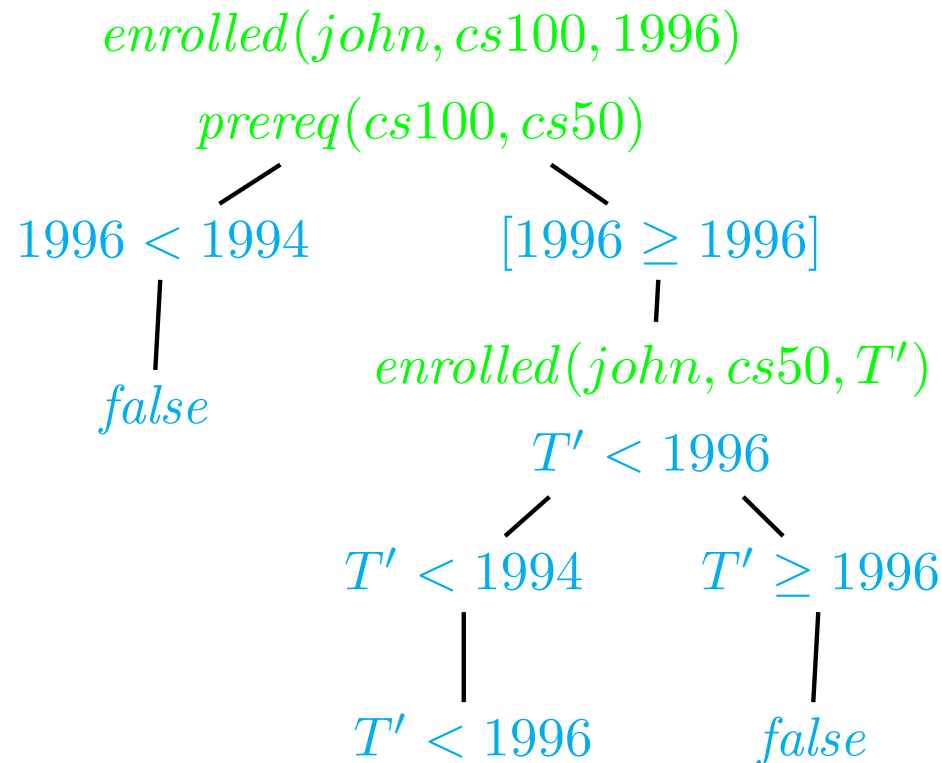
Bottom-up evaluation with disjunction, existential variables, and constraints

$enrolled(S, C, T) \wedge prereq(C, C') \Rightarrow enrolled(S, C', T') \wedge T' < T.$
 $enrolled(john, C, T) \Rightarrow T < 1994 \vee T \geq 1996.$

CHR[∨] and Automated Reasoning

Bottom-up evaluation with disjunction, existential variables, and constraints

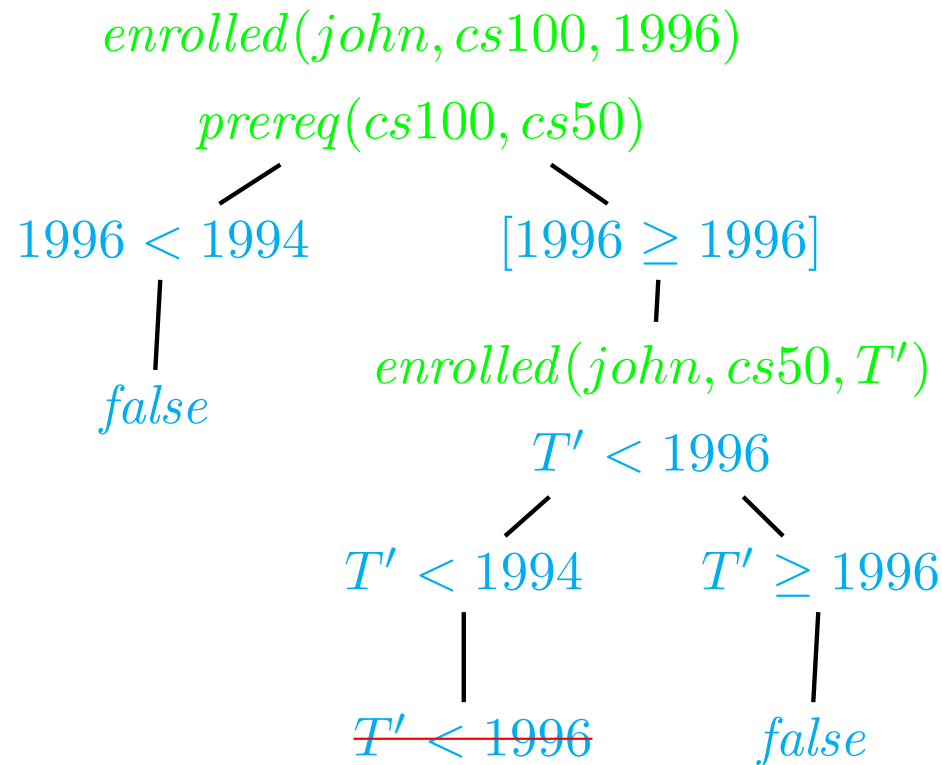
$enrolled(S, C, T) \wedge prereq(C, C') \Rightarrow enrolled(S, C', T') \wedge T' < T.$
 $enrolled(john, C, T) \Rightarrow T < 1994 \vee T \geq 1996.$



CHR[∨] and Automated Reasoning

Bottom-up evaluation with disjunction, existential variables, and constraints

$enrolled(S, C, T) \wedge prereq(C, C') \Rightarrow enrolled(S, C', T') \wedge T' < T.$
 $enrolled(john, C, T) \Rightarrow T < 1994 \vee T \geq 1996.$

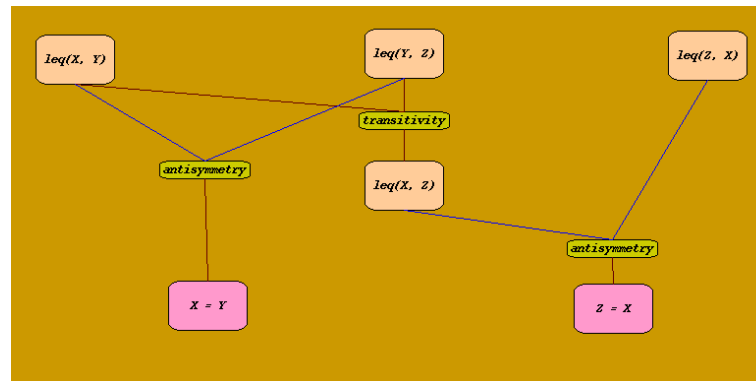


CHR Implementations

- Eclipse Prolog, YAP Prolog, Sicstus Prolog (CHR online)

`www.pms.informatik.uni-muenchen.de/~webchr/`

- Haskell
- JACK (Java Constraint Kit) [WFLP'01, WLPE'01]
 - JCHR: Java Constraint Handling Rules
 - JASE: Java Abstract Search Engine
 - VisualCHR: An interactive tool to visualize JCHR computations



Overview

- Introduction of CHR: Syntax and Semantics
- Reasoning about CHR: Program Analysis
- Reasoning with CHR: Implementation
- Reasoning for CHR: Automatic Generation of Constraint Solvers

Motivation

Writing a constraint solver is in general a difficult task

Motivation

Writing a constraint solver is in general a difficult task

Goal: Automatic generation of constraint solving algorithms in form of rules, where the user:

- gives extensional or intensional definitions of the constraints
- specifies the admissible syntactic form of the rules

Automatic Generation of Rule-based Constraint Solvers

Automatic Generation of Rule-based Constraint Solvers

Step 1: Generation of propagation rules

[CP'00, CP'01, ICTAI'01, IJAIT'02]

$$\text{and}(0, Y, Z) \Rightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{neg}(X, Y) \Rightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{or}(Z, Y, Z1) \Rightarrow Y=Z1.$$

Automatic Generation of Rule-based Constraint Solvers

Step 1: Generation of propagation rules

[CP'00, CP'01, ICTAI'01, IJAIT'02]

$$\text{and}(0, Y, Z) \Rightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{neg}(X, Y) \Rightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{or}(Z, Y, Z1) \Rightarrow Y=Z1.$$

Step 2: Transformation of propagation rules into simplification rules

[PPDP'01]

Automatic Generation of Rule-based Constraint Solvers

Step 1: Generation of propagation rules

[CP'00, CP'01, ICTAI'01, IJAIT'02]

$$\text{and}(0, Y, Z) \Rightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{neg}(X, Y) \Rightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{or}(Z, Y, Z1) \Rightarrow Y=Z1.$$

Step 2: Transformation of propagation rules into simplification rules

[PPDP'01]

$$\text{and}(0, Y, Z) \Leftrightarrow Z=0.$$

Automatic Generation of Rule-based Constraint Solvers

Step 1: Generation of propagation rules

[CP'00, CP'01, ICTAI'01, IJAIT'02]

$$\text{and}(0, Y, Z) \Rightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{neg}(X, Y) \Rightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{or}(Z, Y, Z1) \Rightarrow Y=Z1.$$

Step 2: Transformation of propagation rules into simplification rules

[PPDP'01]

$$\text{and}(0, Y, Z) \Leftrightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{neg}(X, Y) \Leftrightarrow \text{neg}(X, Y), Z=0.$$

Automatic Generation of Rule-based Constraint Solvers

Step 1: Generation of propagation rules

[CP'00, CP'01, ICTAI'01, IJAIT'02]

$$\text{and}(0, Y, Z) \Rightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{neg}(X, Y) \Rightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{or}(Z, Y, Z1) \Rightarrow Y=Z1.$$

Step 2: Transformation of propagation rules into simplification rules

[PPDP'01]

$$\text{and}(0, Y, Z) \Leftrightarrow Z=0.$$

$$\text{and}(X, Y, Z), \text{neg}(X, Y) \Leftrightarrow \text{neg}(X, Y), Z=0.$$

$$\text{and}(X, Y, Z), \text{or}(Z, Y, Z1) \Rightarrow Y=Z1.$$

Generation of Propagation Rules

Syntax

$$C_L \Rightarrow C_R$$

$$C_L \Rightarrow \textit{false}$$

where C_L and C_R are sets of atomic constraints

PROPMINER Algorithm

[CP'00]

INPUT

- *Base*: constraints for which rules have to be generated
- $Cand_L$: candidate constraints for lhs
- $Cand_R$: candidate constraints for rhs
- Definition of *Base* and solvers for $Cand_L$ and $Cand_R$

PROPMINER Algorithm

[CP'00]

INPUT

- *Base*: constraints for which rules have to be generated
- $Cand_L$: candidate constraints for lhs
- $Cand_R$: candidate constraints for rhs
- Definition of *Base* and solvers for $Cand_L$ and $Cand_R$

ALGORITHM

$\forall C_L$ determine C_R as follows:

if $C_L \models \perp$, then $C_L \Rightarrow false$

else $C_R = \{C_i \in Cand_R \mid C_L \models C_i\}$

if $C_R \neq \emptyset$, then $C_L \Rightarrow C_R$

Example: Boolean Conjunction

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

$Base = \{and(X, Y, Z)\}$

$Cand_L = Cand_R = \{X=0, X=1, \dots, Z=1, X=Y, X=Z, Y=Z\}$

Example: Boolean Conjunction

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

$$Base = \{and(X, Y, Z)\}$$

$$Cand_L = Cand_R = \{X=0, X=1, \dots, Z=1, X=Y, X=Z, Y=Z\}$$

$$and(X, Y, Z)$$

Example: Boolean Conjunction

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

$$Base = \{and(X, Y, Z)\}$$

$$Cand_L = Cand_R = \{X=0, X=1, \dots, Z=1, X=Y, X=Z, Y=Z\}$$

$$and(X, Y, Z)$$

$$and(X, Y, Z), X=0$$

Example: Boolean Conjunction

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

$$Base = \{and(X, Y, Z)\}$$

$$Cand_L = Cand_R = \{X=0, X=1, \dots, Z=1, X=Y, X=Z, Y=Z\}$$

$$and(X, Y, Z)$$

$$and(X, Y, Z), X=0$$

$$and(X, Y, Z), X=0, Y=0$$

Example: Boolean Conjunction

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

$$Base = \{and(X, Y, Z)\}$$

$$Cand_L = Cand_R = \{X=0, X=1, \dots, Z=1, X=Y, X=Z, Y=Z\}$$

$$and(X, Y, Z)$$

$$and(X, Y, Z), X=0$$

$$and(X, Y, Z), X=0, Y=0$$

...

$$and(X, Y, Z), X=Y$$

...

Example: Boolean Conjunction

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

$$Base = \{and(X, Y, Z)\}$$

$$Cand_L = Cand_R = \{X=0, X=1, \dots, Z=1, X=Y, X=Z, Y=Z\}$$

$$\cancel{and(X, Y, Z)}$$

$$and(X, Y, Z), X=0$$

$$and(X, Y, Z), X=0, Y=0$$

...

$$and(X, Y, Z), X=Y$$

...

Example: Boolean Conjunction

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

$$Base = \{and(X, Y, Z)\}$$

$$Cand_L = Cand_R = \{X=0, X=1, \dots, Z=1, X=Y, X=Z, Y=Z\}$$

~~$and(X, Y, Z)$~~

$and(X, Y, Z), X=0 \Rightarrow Z=0, X=Z.$

$and(X, Y, Z), X=0, Y=0$

...

$and(X, Y, Z), X=Y$

...

Example: Boolean Conjunction

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

$$Base = \{and(X, Y, Z)\}$$

$$Cand_L = Cand_R = \{X=0, X=1, \dots, Z=1, X=Y, X=Z, Y=Z\}$$

~~$and(X, Y, Z)$~~

$$and(X, Y, Z), X=0 \Rightarrow Z=0, X=Z.$$

$$and(X, Y, Z), X=0, Y=0 \Rightarrow Z=0, X=Z, Y=Z.$$

...

$$and(X, Y, Z), X=Y$$

...

Example: Boolean Conjunction

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

$$Base = \{and(X, Y, Z)\}$$

$$Cand_L = Cand_R = \{X=0, X=1, \dots, Z=1, X=Y, X=Z, Y=Z\}$$

~~$and(X, Y, Z)$~~

$$and(X, Y, Z), X=0 \Rightarrow Z=0, X=Z.$$

$$and(X, Y, Z), X=0, Y=0 \Rightarrow Z=0, X=Z, Y=Z.$$

...

$$and(X, Y, Z), X=Y \Rightarrow X=Z, Y=Z.$$

...

PROPMINER Algorithm

[CP'00]

INPUT

- *Base*: constraints for which rules have to be generated
- $Cand_L$: candidate constraints for the left hand side
- $Cand_R$: candidate constraints for the right hand side
- Definition of *Base* and solvers for $Cand_L$ and $Cand_R$

ALGORITHM

$\forall C_L$ determine C_R as follows:

if $C_L \models \perp$, then $C_L \Rightarrow false$

else $C_R = \{C_i \in Cand_R \mid C_L \models C_i\}$

if $C_R \neq \emptyset$, then $C_L \Rightarrow C_R$

PROPMINER Algorithm

[CP'00]

INPUT

- *Base*: constraints for which rules have to be generated
- $Cand_L$: candidate constraints for the left hand side
- $Cand_R$: candidate constraints for the right hand side
- Definition of *Base* and solvers for $Cand_L$ and $Cand_R$

ALGORITHM

$\forall C_L$ determine C_R as follows:

if $C_L \models \perp$, then $C_L \Rightarrow false$

$neg(X, X) \Rightarrow false$

else $C_R = \{C_i \in Cand_R \mid C_L \models C_i\}$

if $C_R \neq \emptyset$, then $C_L \Rightarrow C_R$

Pruning Strategies (C_L from general to specific)

1. If a rule $C_L \Rightarrow false$ is generated then do not consider any superset of C_L .

Pruning Strategies (C_L from general to specific)

1. If a rule $C_L \Rightarrow false$ is generated then do not consider any superset of C_L .
2. If a rule $C_L \Rightarrow C_R$ is generated then do not consider any C such that $C_L \subset C$ and $C \cap C_R \neq \emptyset$.

Pruning Strategies (C_L from general to specific)

1. If a rule $C_L \Rightarrow false$ is generated then do not consider any superset of C_L .
2. If a rule $C_L \Rightarrow C_R$ is generated then do not consider any C such that $C_L \subset C$ and $C \cap C_R \neq \emptyset$.

Example:

$$and(X, Y, Z), neg(A, B), A=X, B=Y \Rightarrow Z=0$$

$$and(X, Y, Z), neg(A, B), A=X, B=Y, B=1, Z=0$$

Pruning Strategies (C_L from general to specific)

1. If a rule $C_L \Rightarrow false$ is generated then do not consider any superset of C_L .
2. If a rule $C_L \Rightarrow C_R$ is generated then do not consider any C such that $C_L \subset C$ and $C \cap C_R \neq \emptyset$.

Example:

$$and(X, Y, Z), neg(A, B), A=X, B=Y \Rightarrow Z=0$$

$$\del{and(X, Y, Z), neg(A, B), A=X, B=Y, B=1, Z=0}$$

Pruning Strategies (C_L from general to specific)

1. If a rule $C_L \Rightarrow false$ is generated then do not consider any superset of C_L .
2. If a rule $C_L \Rightarrow C_R$ is generated then do not consider any C such that $C_L \subset C$ and $C \cap C_R \neq \emptyset$.

Example:

$$and(X, Y, Z), neg(A, B), A=X, B=Y \Rightarrow Z=0$$

$$~~and(X, Y, Z), neg(A, B), A=X, B=Y, B=1, Z=0~~$$

$$and(X, Y, Z), neg(A, B), A=X, B=Y, B=1$$

Generation of Propagation Rules

Pruning Strategies (C_L from general to specific)

1. If a rule $C_L \Rightarrow false$ is generated then do not consider any superset of C_L .
2. If a rule $C_L \Rightarrow C_R$ is generated then do not consider any C such that $C_L \subset C$ and $C \cap C_R \neq \emptyset$.

Example:

$and(X, Y, Z), neg(A, B), A=X, B=Y \Rightarrow Z=0$

~~$and(X, Y, Z), neg(A, B), A=X, B=Y, B=1, Z=0$~~

$and(X, Y, Z), neg(A, B), A=X, B=Y, B=1$ leads to

$and(X, Y, Z), neg(A, B), A=X, B=Y, B=1 \Rightarrow Z=0, A=0, X=0, Y=1.$

Applications [CP'00]

- **Boolean Constraints:** > 100 rules for $\neg, \wedge, \vee, \oplus$
- **Multi-Valued Logics**
- **Temporal Reasoning** (Allen's Interval Approach):
489 rules for composition
- **Spatial Reasoning** (Region Connection Calculus RCC-8):
178 rules for composition

Example: Multi-Valued Logics

X	Y	$X \equiv Y$
t	t	t
t	f	f
t	u	u
f	t	f
f	f	t
f	u	u
u	t	u
u	f	u
u	u	u

$$eq3val(X, X, X) \Rightarrow X \neq f.$$

$$eq3val(X, Y, t) \Rightarrow X \neq u, X = Y.$$

$$eq3val(X, f, X) \Rightarrow X = u.$$

...

Example: Allen's Interval

13 × 13 table defining a “composition” constraint:

$$\text{allenComp}(R_1, R_2, R_3) \leftrightarrow (R_1(X, Y) \wedge R_2(Y, Z) \rightarrow R_3(X, Z)),$$

where R_1, R_2, R_3 are primitive interval relations.

489 generated rules

$$\text{allenComp}(R, R, R) \Rightarrow R \neq m, R \neq mi.$$

$$\text{allenComp}(R, R, e) \Rightarrow R = e.$$

$$\text{allenComp}(o, b, R) \Rightarrow R = b.$$

...

Specific Applications

- **Crossword Compilation:**

w6(b,e,t,t,e,r). w5(b,r,a,k,e). w4(b,u,m,p).

w6(c,a,n,n,o,n). w5(b,l,o,k,e). w4(p,l,a,y).

w6(w,e,a,l,t,h). w5(s,t,e,a,m). w4(f,r,e,e).

w6(d,e,a,r,t,h). w5(c,r,e,a,m). w4(s,t,o,p).

w5(p,a,t,c,h).

w5(p,i,t,c,h).

	1	2	3	4	5	6
A	■	□	□	□	□	■
B	■	□	■	■	□	■
C	□	□	□	□	□	□
D	■	□	■	■	□	■
E	■	□	■	■	□	■

- **Security Policies:** Stuckey and Sulzmann

Generation of Propagation Rules

For Constraints Defined Intensionally

Having the definition of minimum:

$$\mathit{min}(A, B, C) \leftarrow A \leq B, C = A.$$

$$\mathit{min}(A, B, C) \leftarrow B \leq A, C = B.$$

Generation of Propagation Rules

For Constraints Defined Intensionally

Having the definition of minimum:

$$\min(A, B, C) \leftarrow A \leq B, C = A.$$

$$\min(A, B, C) \leftarrow B \leq A, C = B.$$

How to automatically generate propagation rules:

$$\min(A, B, C) \Rightarrow C \leq A, C \leq B.$$

$$\min(A, A, C) \Rightarrow A = C.$$

$$\min(A, B, C), C \neq B \Rightarrow C = A.$$

$$\min(A, B, C), C \neq A \Rightarrow C = B.$$

$$\min(A, B, C), B \leq A \Rightarrow C = B.$$

$$\min(A, B, C), A \leq B \Rightarrow C = A.$$

Generation of Propagation Rules

Definition of minimum:

$$\min(A, B, C) \leftarrow A \leq B, C = A.$$

$$\min(A, B, C) \leftarrow B \leq A, C = B.$$

$$\text{Base} = \{\min(A, B, C)\}$$

$$\text{Cand}_L = \{A \leq B, \dots, C \leq B, A = B, \dots, B = C, A \neq B, \dots, B \neq C\}$$

Generation of Propagation Rules

Definition of minimum:

$$\min(A, B, C) \leftarrow A \leq B, C = A.$$

$$\min(A, B, C) \leftarrow B \leq A, C = B.$$

$$\text{Base} = \{\min(A, B, C)\}$$

$$\text{Cand}_L = \{A \leq B, \dots, C \leq B, A = B, \dots, B = C, A \neq B, \dots, B \neq C\}$$

Goal: $\min(A, B, C), A \leq B$

Generation of Propagation Rules

Definition of minimum:

$$\min(A, B, C) \leftarrow A \leq B, C = A.$$

$$\min(A, B, C) \leftarrow B \leq A, C = B.$$

$$\text{Base} = \{\min(A, B, C)\}$$

$$\text{Cand}_L = \{A \leq B, \dots, C \leq B, A = B, \dots, B = C, A \neq B, \dots, B \neq C\}$$

Goal: $\min(A, B, C)$, $A \leq B$

Answers: $A \leq B$, $A = C$ and $A = B$, $A = C$

Generation of Propagation Rules

Definition of minimum:

$$\min(A, B, C) \leftarrow A \leq B, C = A.$$

$$\min(A, B, C) \leftarrow B \leq A, C = B.$$

$$\text{Base} = \{\min(A, B, C)\}$$

$$\text{Cand}_L = \{A \leq B, \dots, C \leq B, A = B, \dots, B = C, A \neq B, \dots, B \neq C\}$$

Goal: $\min(A, B, C), A \leq B$

Answers: $A \leq B, A = C$ and $A = B, A = C$

Least general generalization (lgg): $A = C$

Generation of Propagation Rules

Definition of minimum:

$$\min(A, B, C) \leftarrow A \leq B, C = A.$$

$$\min(A, B, C) \leftarrow B \leq A, C = B.$$

$$\text{Base} = \{\min(A, B, C)\}$$

$$\text{Cand}_L = \{A \leq B, \dots, C \leq B, A = B, \dots, B = C, A \neq B, \dots, B \neq C\}$$

Goal: $\min(A, B, C), A \leq B$

Answers: $A \leq B, A = C$ and $A = B, A = C$

Least general generalization (lgg): $A = C$

The algorithm generates the rule: $\min(A, B, C), A \leq B \Rightarrow A = C$.

Generation of Propagation Rules

INPUT

- *Base*: constraints for which rules have to be generated
- *Cand_L*: candidate constraints for the left hand side
- solver (eventually not complete) for primitive constraints
- a CLP program *P* defining the constraints of interest

ALGORITHM [CP'01, IJAIT'02]

generate each possible left hand side C_L wrt. *Base* and *Cand_L*

for each C_L determine C_R as follows

let \mathcal{A} be the set of answers for the goal C_L wrt. *P*

if $\mathcal{A} = \emptyset$ then $C_L \Rightarrow false$

if \mathcal{A} is finite then compute $C_R := lgg(\mathcal{A})$ $lgg \rightarrow$ [plotkin70]

if $C_R \neq \emptyset$ then $C_L \Rightarrow C_R$

Generation of Propagation Rules

Limit of syntactic *lgg*

$$\min(A, B, C) \leftarrow A \leq B, C = A.$$

$$\min(A, B, C) \leftarrow B \leq A, C = B.$$

Goal: $\min(A, B, C)$

Answers:

$$A \leq B, A = C$$

$$B \leq A, B = C$$

lgg ?

Generation of Propagation Rules

Limit of syntactic *lgg*

$$\min(A, B, C) \leftarrow A \leq B, C = A.$$

$$\min(A, B, C) \leftarrow B \leq A, C = B.$$

Goal: $\min(A, B, C)$

Answers:

$$A \leq B, A = C, A \leq C, C \leq A, C \leq B$$

$$B \leq A, B = C, B \leq C, C \leq B, C \leq A$$

lgg ?

Generation of Propagation Rules

Limit of syntactic *lgg*

$$\min(A, B, C) \leftarrow A \leq B, C = A.$$

$$\min(A, B, C) \leftarrow B \leq A, C = B.$$

Goal: $\min(A, B, C)$

Answers:

$$A \leq B, A = C, A \leq C, C \leq A, C \leq B$$

$$B \leq A, B = C, B \leq C, C \leq B, C \leq A$$

lgg ?

$$C \leq A, C \leq B$$

Generation of Propagation Rules

Limit of syntactic *lgg*

$$\min(A, B, C) \leftarrow A \leq B, C = A.$$

$$\min(A, B, C) \leftarrow B \leq A, C = B.$$

Goal: $\min(A, B, C)$

Answers:

$$A \leq B, A = C, A \leq C, C \leq A, C \leq B$$

$$B \leq A, B = C, B \leq C, C \leq B, C \leq A$$

lgg ?

$$C \leq A, C \leq B$$

then the algorithm generates the rule

$$\min(A, B, C) \Rightarrow C \leq A, C \leq B.$$

Interaction of *min* and *max*

Using rules for *min* and rules for *max*, many redundant rules are discarded.

10 propagation rules specific to the interaction of *min* and *max* are generated.

Examples

$$\min(A, B, C), \max(D, E, F), C \neq E, C \neq D \Rightarrow F \neq C.$$

$$\min(A, B, C), \max(D, E, F), B \neq D, A \neq D \Rightarrow D \neq C.$$

$$\min(A, B, C), \max(D, E, F), C \neq E, B \neq D, A \neq F \Rightarrow F \neq C.$$

$$\min(A, B, C), \max(D, E, F), C \neq D, B \neq F, A \neq E \Rightarrow F \neq C.$$

Generation of Propagation Rules for Recursive Definitions

- bound the depth of the resolution
- prefer a resolution based on the OLDT scheme

$$\text{append}(X, Y, Z) \leftarrow X=[] \wedge Y=Z.$$

$$\text{append}(X, Y, Z) \leftarrow X=[H|X1] \wedge Z=[H|Z1] \wedge \text{append}(X1, Y, Z1).$$

Example of rules (with a bounded resolution depth)

$$\text{append}(A, B, C) \wedge A=B \wedge C=[D] \Rightarrow \text{false}.$$

$$\text{append}(A, B, C) \wedge B=C \wedge C=[D] \Rightarrow A=[].$$

$$\text{append}(A, B, C) \wedge C=[] \Rightarrow B=[] \wedge A=[].$$

$$\text{append}(A, B, C) \wedge A=[] \Rightarrow B=C.$$

Motivation

- Propagation rules do not rewrite constraints but add new ones
- Simplification rules remove constraints from the constraint store

Motivation

- Propagation rules do not rewrite constraints but add new ones
- Simplification rules remove constraints from the constraint store

Removing constraints

- allows saving of space
- decreases the cost of constraint solving

Motivation

- Propagation rules do not rewrite constraints but add new ones
- Simplification rules remove constraints from the constraint store

Removing constraints

- allows saving of space
- decreases the cost of constraint solving

Problem: Find criteria to transform some propagation rules into simplification rules

Semantical Criterion: Same Solutions

Example: $and(X, Y, Z), neg(X, Y) \Rightarrow Z=0$

Semantical Criterion: Same Solutions

Example: $and(X, Y, Z), neg(X, Y) \Rightarrow Z=0$

1. $and(X, Y, Z), neg(X, Y) \Leftrightarrow Z=0$

Semantical Criterion: Same Solutions

Example: $and(X, Y, Z), neg(X, Y) \Rightarrow Z=0$

1. $and(X, Y, Z), neg(X, Y) \Leftrightarrow Z=0$

Counterexample: $X=0, Y=0, Z=0$

Semantical Criterion: Same Solutions

Example: $and(X, Y, Z), neg(X, Y) \Rightarrow Z=0$

1. $and(X, Y, Z), neg(X, Y) \Leftrightarrow Z=0$

Counterexample: $X=0, Y=0, Z=0$

2. $and(X, Y, Z), neg(X, Y) \Leftrightarrow and(X, Y, Z), Z=0$

Semantical Criterion: Same Solutions

Example: $and(X, Y, Z), neg(X, Y) \Rightarrow Z=0$

1. $and(X, Y, Z), neg(X, Y) \Leftrightarrow Z=0$

Counterexample: $X=0, Y=0, Z=0$

2. $and(X, Y, Z), neg(X, Y) \Leftrightarrow and(X, Y, Z), Z=0$

Counterexample: $X=0, Y=0, Z=0$

Semantical Criterion: Same Solutions

Example: $and(X, Y, Z), neg(X, Y) \Rightarrow Z=0$

1. $and(X, Y, Z), neg(X, Y) \Leftrightarrow Z=0$

Counterexample: $X=0, Y=0, Z=0$

2. $and(X, Y, Z), neg(X, Y) \Leftrightarrow and(X, Y, Z), Z=0$

Counterexample: $X=0, Y=0, Z=0$

3. $and(X, Y, Z), neg(X, Y) \Leftrightarrow neg(X, Y), Z=0$

SIMPMINER Algorithm

INPUT: A set P of propagation rules

OUTPUT: A set P' consisting of propagation and simplification rules

ALGORITHM:

$P' := \emptyset$

for each rule R of the form $H \Rightarrow B$ in P **do**

Find $R' := H \Leftrightarrow B \wedge C$ with $C \subset H$ such that
 H and $B \wedge C$ have the same solutions

If R' exists

then $P' := P' \cup \{R'\}$

else $P' := P' \cup \{R\}$

Syntactical Criterion: Confluence

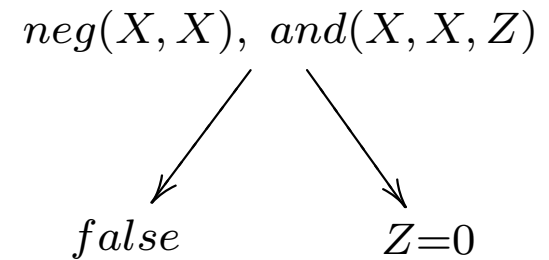
Example: $and(X, Y, Z), neg(X, Y) \Rightarrow Z=0$

Syntactical Criterion: Confluence

Example: $and(X, Y, Z), neg(X, Y) \Rightarrow Z=0$

1.

$and(X, Y, Z), neg(X, Y) \Leftrightarrow Z=0$



Syntactical Criterion: Confluence

Example: $and(X, Y, Z), neg(X, Y) \Rightarrow Z=0$

1.

$$and(X, Y, Z), neg(X, Y) \Leftrightarrow Z=0$$

2.

$$and(X, Y, Z), neg(X, Y) \Leftrightarrow and(X, Y, Z), Z=0$$

$neg(X, X), and(X, X, Z)$

$false$

$Z=0$

$neg(X, X), and(X, X, Z)$

$false$

$X=0, Z=0$

Syntactical Criterion: Confluence

Example: $and(X, Y, Z), neg(X, Y) \Rightarrow Z=0$

1.

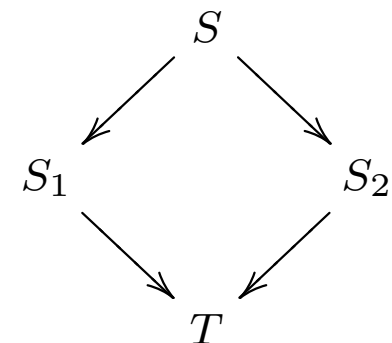
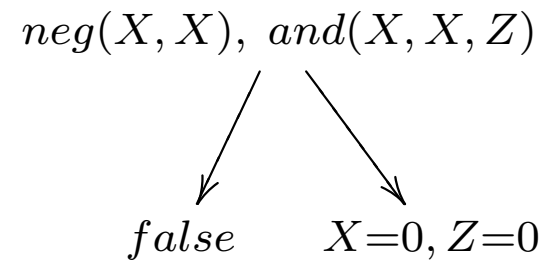
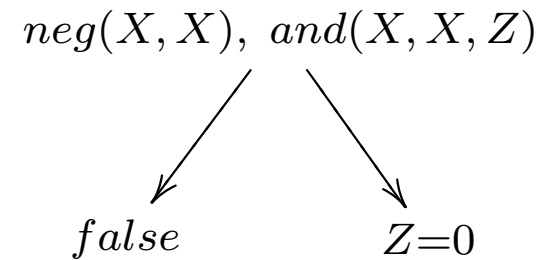
$$and(X, Y, Z), neg(X, Y) \Leftrightarrow Z=0$$

2.

$$and(X, Y, Z), neg(X, Y) \Leftrightarrow and(X, Y, Z), Z=0$$

3.

$$and(X, Y, Z), neg(X, Y) \Leftrightarrow neg(X, Y), Z=0$$



SIMPMINER Algorithm

[PPDP'01]

INPUT: A set P consisting of propagation rules

OUTPUT: A set P' consisting of propagation and simplification rules

ALGORITHM:

$P' := P$

for each rule R of the form $H \Rightarrow B$ in P **do**

Find $R' := H \Leftrightarrow B \wedge C$ with $C \subset H$ such that
 $(P' \setminus \{R\}) \cup \{R'\}$ is terminating and confluent.

If R' exists

then $P' := (P' \setminus \{R\}) \cup \{R'\}$

Generation of Simplification Rules

Example

$$\begin{aligned} \min(A, B, C) &\Rightarrow C \leq A, C \leq B. \\ \min(A, A, C) &\Rightarrow A = C. \\ \min(A, B, C), C \neq B &\Rightarrow C = A. \\ \min(A, B, C), C \neq A &\Rightarrow C = B. \\ \min(A, B, C), B \leq A &\Rightarrow C = B. \\ \min(A, B, C), A \leq B &\Rightarrow C = A. \end{aligned}$$

is transformed by SIMPMINER into

$$\begin{aligned} \min(A, B, C) &\Rightarrow C \leq A, C \leq B. \\ \min(A, A, C) &\Leftrightarrow A = C. \\ \min(A, B, C), C \neq B &\Rightarrow C = A. \\ \min(A, B, C), C \neq A &\Rightarrow C = B. \\ \min(A, B, C), B \leq A &\Leftrightarrow C = B, B \leq A. \\ \min(A, B, C), A \leq B &\Leftrightarrow C = A, A \leq B. \end{aligned}$$

Automatic Test-Pattern Generation (ATPG)

CLP Approach proposed by Van Hentenryck et al: **Six-valued logic**

- Propagation rules with one atom in the lhs (77 rules)
- Propagation rules with one or two atoms in the lhs (621 rules)
 - the size of the search space is reduced
 - overhead in terms of execution time
- 308 propagation rules have been transformed into simplification rules
 - execution time is reduced by more than 50%

[PPDP'01]

Constraint Handling Rules

- Declarative language for constraint programming
- Executable specification and rapid prototyping
- Computing with incomplete information
- Good theoretical properties
- Implementation and libraries available
- Semi-automatic generation