Reasoning with, about and for Constraint Handling Rules

Slim Abdennadher
LMU München

Christophe Rigotti
INSA Lyon

22.7.2002


**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:
  
  \[ x \in \{0, 1, \ldots, 9\} \land x \geq 5 \land prime(x) \]

- **Constraint solving** reduce search space:
  
  \[ x \in \{0, 1, \ldots, 9\} \land x \geq 5 \rightarrow x \in \{5, 6, 7, 8, 9\} \]

  \[ x \in \{5, 6, 7, 8, 9\} \land prime(x) \rightarrow x \in \{5, 7\} \]

- **Search**:

  \[ x \in \{5, 7\} \]

  \[
  \begin{array}{c}
  x=5 \\
  \end{array} \quad \begin{array}{c}
  x=7 \\
  \end{array}
  \]
**Crypto-arithmetic Problem**

\[
\begin{array}{ccccc}
S & E & N & D \\
+ & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}
\]

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

- \([S,E,N,D,M,O,R,Y]\) in 0..9,
- \(S 
eq 0, M 
eq 0,\)
- 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
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
A concurrent constraint language with ask and tell (50+ applications) for computational logic and more...

**Extensions**: Disjunction/Search, Dynamic Constraints, ...
CHR: Introductory Example \( \leq \)

\[
\begin{align*}
X & \leq X & \Leftrightarrow & \text{true} & \text{(reflexivity)} \\
X & \leq Y & \land & Y \leq X & \Leftrightarrow & X = Y & \text{(antisymmetry)} \\
X & \leq Y & \land & Y \leq Z & \Rightarrow & X \leq Z & \text{(transitivity)}
\end{align*}
\]

\[
\begin{align*}
A & \leq B & \land & B \leq C & \land & C \leq A \\
& \downarrow & & & & & & & \text{(transitivity)} \\
A & \leq B & \land & B \leq C & \land & C \leq A & \land & A \leq C \\
& \downarrow & & & & & & & \text{(antisymmetry)} \\
A & \leq B & \land & B \leq C & \land & A = C \\
& \downarrow & & & & & & & \text{(Black-box solver)} \\
A & \leq B & \land & B \leq A & \land & A = C \\
& \downarrow & & & & & & & \text{(antisymmetry)} \\
A & = B & \land & A = C
\end{align*}
\]
Simplification rule: \[ H \iff C \mid B \quad \forall \bar{x} (C \rightarrow (H \iff \exists y B)) \]

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

- \( H \): 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 \(CT \models G_{\text{built}} \rightarrow \exists \bar{x}(H=H' \land C)\)
then \(H' \land G \leftrightarrow G \land H=H' \land B\)

Propagate

If \((H \Rightarrow C \mid B)\) is a fresh variant of a rule with variables \(\bar{x}\)
and \(CT \models G_{\text{built}} \rightarrow \exists \bar{x}(H=H' \land C)\)
then \(H' \land G \leftrightarrow H' \land G \land H=H' \land B\)
Anytime Algorithm

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

\[
\begin{align*}
A \leq B \land B \leq C \land C \leq A \\
\quad \downarrow \\
A \leq B \land B \leq C \land C \leq A \land A \leq C \\
\quad \downarrow \\
A \leq B \land B \leq C \land A = C \\
\quad \downarrow \\
A \leq B \land B \leq A \land A = C \\
\quad \downarrow \\
A = B \land A = C
\end{align*}
\] (transitivity)

(antisymmetry)

(Black-box solver)

(antisymmetry)
Operational Properties

Monotonicity and Incrementality

If \( G \rightarrow G' \)
then \( G \land C \rightarrow G' \land C \)

Online Algorithm
The complete input is initially unknown.
The input data arrives during computation.
No recomputation from scratch necessary.

\[
\begin{align*}
A \leq B & \land B \leq C \land C \leq A \\
\downarrow & \\
A \leq B & \land B \leq C \land A \leq C \land C \leq A \\
\downarrow & \\
A \leq B & \land B \leq C \land A = C \\
\downarrow & \\
\cdots & 
\end{align*}
\]

(transitivity)

(antisymmetry)
Concurrency

If \( A \mapsto B \) and \( C \mapsto D \), then \( A \land C \mapsto B \land D \)

\[
\begin{align*}
&\overbrace{A \leq B \land B \leq C} \land \overbrace{C \leq D \land D \leq A} \\
&\downarrow \quad \downarrow \\
&\overbrace{A \leq B \land B \leq C \land A \leq C} \land \overbrace{C \leq D \land D \leq A \land C \leq A} \\
&\downarrow \\
&\ldots
\end{align*}
\]
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]
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:** \( H \land C \)

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.

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

\[
X \leq X \iff true \quad \text{(reflexivity)}
\]
\[
X \leq Y \land Y \leq X \iff X = Y \quad \text{(antisymmetry)}
\]

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

\[ X \leq Y \land Y \leq X \iff X = Y \quad \text{(antisymmetry)} \]
\[ X \leq Y \land Y < X \iff false \quad \text{(inconsistency)} \]

\[ A \leq B \land B \leq A \land B < A \]

antisymmetry \quad \text{inconsistency}

\[ A = B \land B < A \]
\[ B \leq A \land false \]

\[ A = B \land A < A \]
\[ false \]

\[ X < X \iff false \quad \text{(irreflexivity)} \]
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 $\text{max}$

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

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

$P_2$ defines $\text{max}$

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

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

$\text{max}(X, Y, Z) \wedge X \geq Y \quad \text{max}(X, Y, Z) \wedge X \geq Y$

\[
\begin{array}{c}
\text{max}(X, Y, Z) \wedge X \geq Y \\
\downarrow P_1 \\
Z = X \wedge X \geq Y
\end{array}
\]

\[
\begin{array}{c}
\text{max}(X, Y, Z) \wedge X \geq Y \\
\downarrow P_2 \\
Z = X \wedge X \geq Y
\end{array}
\]
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**: $\neg$, $\land$, $\lor$, $\oplus$, $\rightarrow$, $\leftrightarrow$

- **Theory**:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$\neg X$</th>
<th>$X \land Y$</th>
<th>$X \lor Y$</th>
<th>$X \oplus Y$</th>
<th>$X \rightarrow Y$</th>
<th>$X \leftrightarrow Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

\[\text{and}(X, X, Z) \iff X = Z.\]
\[\text{and}(X, Y, 1) \iff X = 1 \land Y = 1.\]
\[\text{and}(X, 1, Z) \iff X = Z.\]
\[\text{and}(X, 0, Z) \iff Z = 0.\]
\[\text{and}(1, Y, Z) \iff Y = Z.\]
\[\text{and}(0, Y, Z) \iff Z = 0.\]
\[\neg(X, X) \iff \text{false}.\]
\[\neg(X, 0) \iff X = 1.\]
\[\neg(X, 1) \iff X = 0.\]
\[\neg(0, Y) \iff Y = 1.\]
\[\neg(1, Y) \iff Y = 0.\]

\[\text{and}(X, Y, Z) \land \neg(X, Y) \iff \neg(X, Y) \land Z = 0.\]
\[\text{and}(X, Y, Z) \land \neg(X, Z) \iff X = 1 \land Y = 0 \land Z = 0.\]

...
Syntactic Unification

- **Rational tree**: (possibly infinite) tree with finite set of subtrees, e.g. $X=f(X)$.

- **Solved normal form**:
  
  - `false` or
  
  - $X_1=t_1 \land \ldots \land X_n=t_n$ ($n \geq 0$) where $X_i$ is different to $X_j$ and $t_j$, if $i \leq j$

- **Theory**:
  
  - **Reflexivity**: $\forall (true \rightarrow x=x)$
  
  - **Symmetry**: $\forall (x=y \rightarrow y=x)$
  
  - **Transitivity**: $\forall (x=y \land y=z \rightarrow x=z)$
  
  - **Compatibility**: $\forall (x_1=y_1 \land \ldots \land x_n=y_n \rightarrow f(x_1, \ldots, x_n)=f(y_1, \ldots, y_n))$
  
  - **Decomposition**: $\forall (f(x_1, \ldots, x_n)=f(y_1, \ldots, y_n) \rightarrow x_1=y_1 \land \ldots \land x_n=y_n)$
  
  - **Contradiction**: $\forall (f(x_1, \ldots, x_n)=g(y_1, \ldots, y_m) \rightarrow false)$ if $f \neq g$ or $n \neq m$
Syntactic Unification: Solver

reflexivity @  X=X ⇔ var(X) | true.
orientation @  T=X ⇔ var(X) ∧ X@<T | X=T.
decomposition @  T1=T2 ⇔ nonvar(T1) ∧ nonvar(T2) |
                     same_functor(T1,T2) ∧
                     same_args(T1,T2).
confrontation @  X=T1 ∧ X=T2 ⇔ var(X) ∧ X@<T1 ∧ T1@=<T2 | T1=T2.

\[ h(Y,f(a),g(X,a))=h(f(U),Y,g(h(Y),U)) \]
\[ \text{decomposition} \]
\[ Y=f(U) ∧ f(a)=Y ∧ g(X,a)=g(h(Y),U) \]
\[ \text{orientation} \]
\[ Y=f(U) ∧ Y=f(a) ∧ g(X,a)=g(h(Y),U) \]
\[ \ldots \]
\[ \text{confrontation} \]
\[ Y=f(U) ∧ U=a ∧ X=h(Y) ∧ U=a \]
\[ \text{decomposition} \]
\[ Y=f(U) ∧ U=a ∧ X=h(Y) \]
\[ \text{decomposition} \]
\[ Y=f(U) ∧ U=a ∧ X=h(Y) \]
Linear Polynomial Equations

- *Equations* of the form \( a_1 X_1 + \ldots + a_n X_n + b = 0. \)

- *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 \( a_1 X_1 + \ldots + a_n X_n + b = 0 \)
  - Make its left-most variable explicit:
    \[ X_1 = -(a_2 X_2 + \ldots + a_n X_n + b)/a_1 \]
  - Replace all other occurrences of \( X_1 \) by \( -(a_2 X_2 + \ldots + a_n X_n + b)/a_1. \)
  - Simplify the resulting equations into allowed constraints (this is always possible).
  - Repeat until solved.
Linear Polynomial Equations: Solver

\[ A_1 \cdot X + P_1 = 0 \land X \cdot P = 0 \iff
\begin{align*}
& \text{find}(A_2 \cdot X, X \cdot P, P_2) \mid \\
& \text{canon}(P_2 - (P_1/A_1) \cdot A_2, P_3) \land \\
& A_1 \cdot X + P_1 = 0 \land P_3 = 0.
\end{align*}
\]

\[ B = 0 \iff \text{number}(B) \mid \text{zero}(B). \]

\[ \begin{align*}
1 \cdot X + 3 \cdot Y + 5 &= 0 \land 3 \cdot X + 2 \cdot Y + 8 = 0 \\
\text{canon}((2 \cdot Y + 8) - ((3 \cdot Y + 5)/1) \cdot 3, P_3) &\% P_3 = -7 \cdot Y + -7 \\
1 \cdot X + 3 \cdot Y + 5 &= 0 \land -7 \cdot Y + -7 = 0 &\% Y = -1 \\
\text{canon}((1 \cdot X + 5) - ((-7)/-7) \cdot 3, P_3') &\% P_3' = 1 \cdot X + 2 \\
1 \cdot X + 2 &= 0 \land -7 \cdot Y + -7 = 0 &\% X = -2
\end{align*} \]
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., \( \neg x \lor y \lor z \) as \( \text{cl}([-x,y,z]) \).

\[
\begin{align*}
\text{empty\_clause} @ & \quad \text{cl}([]) \iff \text{false}. \\
\text{tautology} @ & \quad \text{cl}(L) \iff \text{in}(+X,L) \land \text{in}(-X,L) | \text{true}. \\
\text{resolution} @ & \quad \text{cl}(L1) \land \text{cl}(L2) \Rightarrow \\
& \quad \text{find}(+X,L1,L3) \land \text{find}(-X,L2,L4) | \\
& \quad \text{merge}(L3,L4,L) \land \\
& \quad \text{cl}(L). \\
\end{align*}
\]
Bottom-up evaluation of logic programs

\[ p(X, Y) \leftarrow e(X, Y). \]
\[ p(X, Y) \leftarrow e(X, Z) \land p(Z, Y). \]

is transformed into

\[ e(X, Y) \implies p(X, Y). \]
\[ e(X, Z) \land p(Z, Y) \implies p(X, Y). \]

\[
\begin{align*}
    e(a, b) &\land e(b, c) \land e(c, d) \\
    &\downarrow \\
    e(a, b) &\land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \\
    &\downarrow \\
    e(a, b) &\land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \\
    &\downarrow \\
    e(a, b) &\land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d)
\end{align*}
\]
Bottom-up evaluation with disjunction, existential variables, and constraints

\[ \text{enrolled}(S, C, T) \land \text{prereq}(C, C') \Rightarrow \text{enrolled}(S, C', T') \land T' < T. \]
\[ \text{enrolled}(\text{john}, C, T) \Rightarrow T < 1994 \lor T \geq 1996. \]

\[ \text{enrolled}(\text{john}, cs100, 1996) \]
\[ \text{prereq}(cs100, cs50) \]
\[ 1996 < 1994 \]
\[ \text{false} \]
\[ [1996 \geq 1996] \]
\[ \text{enrolled}(\text{john}, cs50, T') \]
\[ T' < 1996 \]
\[ T' < 1994 \]
\[ T' \geq 1996 \]
\[ T' < 1996 \]
\[ \text{false} \]
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

**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
Step 1: Generation of propagation rules
[CP’00, CP’01, ICTAI’01, IJAIT’02]

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

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

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

Step 2: Transformation of propagation rules into simplification rules
[PPDP’01]

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

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

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

\[ C_L \Rightarrow C_R \]
\[ C_L \Rightarrow false \]

where \( C_L \) and \( C_R \) are sets of atomic constraints
**Generation of Propagation Rules**

**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**

\[ \forall C_L \text{ determine } C_R \text{ as follows:} \]

\[ \text{if } C_L \models \bot \text{ , then } C_L \Rightarrow false \]

\[ \text{else } C_R = \{ C_i \in \text{Cand}_R \mid C_L \models C_i \} \]

\[ \text{if } C_R \neq \emptyset , \text{ then } C_L \Rightarrow C_R \]
Example: Boolean Conjunction

<table>
<thead>
<tr>
<th>( X )</th>
<th>( Y )</th>
<th>( X \land Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ \text{Base} = \{ \text{and}(X, Y, Z) \} \]

\[ \text{Cand}_L = \text{Cand}_R = \{ X=0, X=1, \ldots, Z=1, X=Y, X=Z, Y=Z \} \]

\[ \text{and}(X, Y, Z) \]
\[ \text{and}(X, Y, Z), \ X=0 \quad \Rightarrow \quad Z=0, \ X=Z. \]
\[ \text{and}(X, Y, Z), \ X=0, \ Y=0 \quad \Rightarrow \quad Z=0, \ X=Z, \ Y=Z. \]
\[ \ldots \]
\[ \text{and}(X, Y, Z), \ X=Y \quad \Rightarrow \quad X=Z, \ Y=Z. \]
\[ \ldots \]
**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**

∀ $C_L$ determine $C_R$ as follows:

if $C_L \models \perp$, then $C_L \Rightarrow \text{false}$

else $C_R = \{ C_i \in \text{Cand}_R \mid C_L \models C_i \}$

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

$\neg (X, X) \Rightarrow \text{false}$
Pruning Strategies \((C_L \text{ from general to specific})\)

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 \(C_L \subseteq C\) and \(C \cap C_R \neq \emptyset\).

Example:

\[
\text{and}(X, Y, Z), \ neg(A, B), \ A=X, \ B=Y \Rightarrow Z=0
\]

\[
\text{and}(X, Y, Z), \ neg(A, B), \ A=X, \ B=Y, \ B=1, \ Z=0
\]

\[
\text{and}(X, Y, Z), \ neg(A, B), \ A=X, \ B=Y, \ B=1 \text{ leads to}
\]

\[
\text{and}(X, Y, Z), \ neg(A, B), \ A=X, \ B=Y, \ B=1 \Rightarrow Z=0, \ A=0, \ X=0, \ Y=1.
\]
Applications [CP’00]

- **Boolean Constraints**: > 100 rules for $\neg$, $\land$, $\lor$, $\oplus$
- **Multi-Valued Logics**
- **Temporal Reasoning** (Allen’s Interval Approach):
  489 rules for composition
- **Spatial Reasoning** (Region Connection Calculus RCC-8):
  178 rules for composition
Generation of Propagation Rules

Example: Multi-Valued Logics

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>X ≡ Y</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>t</td>
<td>u</td>
<td>u</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>t</td>
</tr>
<tr>
<td>f</td>
<td>u</td>
<td>u</td>
</tr>
<tr>
<td>u</td>
<td>t</td>
<td>u</td>
</tr>
<tr>
<td>u</td>
<td>f</td>
<td>u</td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
</tbody>
</table>

\[ eq3val(X, X, X) \Rightarrow X \neq f. \]
\[ eq3val(X, Y, t) \Rightarrow X \neq u, X = Y. \]
\[ eq3val(X, f, X) \Rightarrow X = u. \]

...
Example: Allen’s Interval

13 × 13 table defining a “composition” constraint:

\[ \text{allenComp}(R_1, R_2, R_3) \leftrightarrow (R_1(X, Y) \land R_2(Y, Z) \rightarrow R_3(X, Z)), \]

where \( R_1, R_2, R_3 \) are primitive interval relations.

489 generated rules

\[ \text{allenComp}(R, R, R) \Rightarrow R \neq m, \ R \neq mi. \]
\[ \text{allenComp}(R, R, e) \Rightarrow R = e. \]
\[ \text{allenComp}(o, b, R) \Rightarrow R = b. \]
\[
\ldots
\]
Specific Applications

- Crossword Compilation:
  
  \[
  \begin{align*}
  w_6(b,e,t,t,e,r) &. w_5(b,r,a,k,e). w_4(b,u,m,p). \\
  w_6(c,a,n,n,o,n) &. w_5(b,l,o,k,e). w_4(p,l,a,y). \\
  w_6(w,e,a,l,t,h) &. w_5(s,t,e,a,m). w_4(f,r,e,e). \\
  w_6(d,e,a,r,t,h) &. w_5(c,r,e,a,m). w_4(s,t,o,p). \\
  w_5(p,a,t,c,h). & \\
  w_5(p,i,t,c,h). &
  \end{align*}
  \]

- Security Policies: Stuckey and Sulzmann
Generation of Propagation Rules

For Constraints Defined Intensionally

Having the definition of minimum:

\[
\min(A, B, C) \leftarrow A \leq B, \quad C = A.
\]
\[
\min(A, B, C) \leftarrow B \leq A, \quad C = B.
\]

How to automatically generate propagation rules:

\[
\min(A, B, C) \Rightarrow C \leq A, \quad C \leq B.
\]
\[
\min(A, A, C) \Rightarrow A = C.
\]
\[
\min(A, B, C), \quad C \neq B \Rightarrow C = A.
\]
\[
\min(A, B, C), \quad C \neq A \Rightarrow C = B.
\]
\[
\min(A, B, C), \quad B \leq A \Rightarrow C = B.
\]
\[
\min(A, B, C), \quad A \leq B \Rightarrow C = A.
\]
Generation of Propagation Rules

Definition of minimum:

\[
\begin{align*}
\min(A, B, C) & \leftarrow A \leq B, \ C = A. \\
\min(A, B, C) & \leftarrow B \leq A, \ C = B.
\end{align*}
\]

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

Cand\(_L\)\(=\) \(\{A \leq B, \ldots, C \leq B, A = B, \ldots, B = C, A \neq B, \ldots, B \neq C\}\)

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 \( A \) be the set of answers for the goal \( C_L \) wrt. \( P \)

if \( A = \emptyset \) then \( C_L \Rightarrow false \)

if \( A \) is finite then compute \( C_R := lgg(A) \quad lgg \rightarrow [plotkin70] \)

if \( C_R \neq \emptyset \) then \( C_L \Rightarrow C_R \)
Generation of Propagation Rules

Limit of syntactic \( lgg \)

\[
\begin{align*}
\min(A, B, C) & \leftarrow A \leq B, \ C = A. \\
\min(A, B, C) & \leftarrow B \leq A, \ C = B.
\end{align*}
\]

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 \textit{min} and \textit{max}

Using rules for \textit{min} and rules for \textit{max}, many redundant rules are discarded. 10 propagation rules specific to the interaction of \textit{min} and \textit{max} are generated.

Examples

\begin{align*}
\text{min}(A, B, C), \text{max}(D, E, F), C \neq E, C \neq D & \Rightarrow F \neq C. \\
\text{min}(A, B, C), \text{max}(D, E, F), B \neq D, A \neq D & \Rightarrow D \neq C. \\
\text{min}(A, B, C), \text{max}(D, E, F), C \neq E, B \neq D, A \neq F & \Rightarrow F \neq C. \\
\text{min}(A, B, C), \text{max}(D, E, F), C \neq D, B \neq F, A \neq E & \Rightarrow F \neq C.
\end{align*}
• bound the depth of the resolution
• prefer a resolution based on the OLDT scheme

\[
\text{append}(X, Y, Z) \leftarrow X=\[] \land Y=Z.
\]
\[
\text{append}(X, Y, Z) \leftarrow X=[H\mid X1] \land Z=[H\mid Z1] \land \text{append}(X1, Y, Z1).
\]

Example of rules (with a bounded resolution depth)

\[
\text{append}(A, B, C) \land A=B \land C=[D] \Rightarrow \text{false}.
\]
\[
\text{append}(A, B, C) \land B=C \land C=[D] \Rightarrow A=\[].
\]
\[
\text{append}(A, B, C) \land C=\[] \Rightarrow B=\[] \land A=\[].
\]
\[
\text{append}(A, B, C) \land A=\[] \Rightarrow B=C.
\]
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: $\text{and}(X, Y, Z), \text{neg}(X, Y) \Rightarrow Z=0$

1. $\text{and}(X, Y, Z), \text{neg}(X, Y) \iff Z=0$
   Counterexample: $X=0, Y=0, Z=0$

2. $\text{and}(X, Y, Z), \text{neg}(X, Y) \iff \text{and}(X, Y, Z), Z=0$
   Counterexample: $X=0, Y=0, Z=0$

3. $\text{and}(X, Y, Z), \text{neg}(X, Y) \iff \text{neg}(X, Y), Z=0$
**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 $R' := H \iff B \land C$ with $C \subset H$ such that $H$ and $B \land C$ have the same solutions

If $R'$ exists

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

else $P' := P' \cup \{R\}$
Generation of Simplification Rules

Syntactical Criterion: Confluence

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

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

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

3. $\text{and}(X, Y, Z), \text{neg}(X, Y) \iff \text{neg}(X, Y), Z=0$
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 $R' := H \Leftrightarrow B \land C$ with $C \subset H$ such that $(P' \setminus \{R\}) \cup \{R'\}$ is terminating and confluent.

If $R'$ exists

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

\[
\begin{align*}
\text{min}(A, B, C) & \Rightarrow C \leq A, \ C \leq B. \\
\text{min}(A, A, C) & \Rightarrow A = C. \\
\text{min}(A, B, C), \ C \neq B & \Rightarrow C = A. \\
\text{min}(A, B, C), \ C \neq A & \Rightarrow C = B. \\
\text{min}(A, B, C), \ B \leq A & \Rightarrow C = B. \\
\text{min}(A, B, C), \ A \leq B & \Rightarrow C = A.
\end{align*}
\]

is transformed by \textsc{SimpMiner} into

\[
\begin{align*}
\text{min}(A, B, C) & \Rightarrow C \leq A, \ C \leq B. \\
\text{min}(A, A, C) & \Leftrightarrow A = C. \\
\text{min}(A, B, C), \ C \neq B & \Rightarrow C = A. \\
\text{min}(A, B, C), \ C \neq A & \Rightarrow C = B. \\
\text{min}(A, B, C), \ B \leq A & \Leftrightarrow C = B, \ B \leq A. \\
\text{min}(A, B, C), \ A \leq B & \Leftrightarrow C = A, \ A \leq B.
\end{align*}
\]
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]
Conclusion

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