Rule-based Constraint Programming: Theory and Practice

Slim Abdennadher

LMU München

18.02.2002
Repaint the Tunisian flag with its two original colors, red and white.
Adjacent regions have different colours.
The star is red.
Constraint Programming

Repaint the Tunisian flag with its two original colors, red and white
Adjacent regions have different colours
The star is red

**Modelling** with Finite Domain Constraints:

\[ [R, C, H, S] \in \{\text{red, white}\} \land R \neq C \land C \neq H \land C \neq S \land S = \text{red} \]
Constraint Programming

Repaint the Tunisian flag with its two original colors, red and white
Adjacent regions have different colours
The star is red

Modelling with Finite Domain Constraints:
\[ [R, C, H, S] \in \{\text{red, white}\} \land R \neq C \land C \neq H \land C \neq S \land S = \text{red} \]

Constraint Propagation:
\[ C \in \{\text{red, white}\} \land S = \text{red} \land C \neq S \rightarrow C = \text{white} \]
\[ [R, H] \in \{\text{red, white}\} \land C = \text{white} \land R \neq C \land C \neq H \rightarrow R = \text{red} \land H = \text{red} \]
Overview

- Automatic Generation of Constraint Solvers
  - Generation of Propagation Rules
  - Generation of Simplification Rules
- Constraint Handling Rules
  - Program Analysis
  - Applications
  - Implementation
- Conclusions
Motivation

Writing a constraint solver is in general a difficult task.

**Example**: Six-valued logic (ATPG, Van Hentenryck et al.): > 2000 cases
Automatic Generation of Constraint Solvers

**Motivation**

Writing a constraint solver is in general a difficult task

**Example:** Six-valued logic (ATPG, Van Hentenryck et al.): > 2000 cases

**Our 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
Motivation

Writing a constraint solver is in general a difficult task

Example: Six-valued logic (ATPG, Van Hentenryck et al.): > 2000 cases

Our 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

These rules

- can be used in rule-based languages (e.g., Frühwirth’s CHR) or
- can be encoded in imperative programming languages.
Example: Boolean Constraints

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X &amp; Y</th>
<th>X \lor Y</th>
<th>\neg X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Example: Boolean Constraints

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X &amp; Y</th>
<th>X ∨ Y</th>
<th>¬X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Simplification Rules

\[ and(0, Y, Z) \quad \Leftrightarrow \quad Z=0. \]
\[ and(X, X, Z) \quad \Leftrightarrow \quad X=Z. \]
\[ neg(X, X) \quad \Leftrightarrow \quad false. \]
\[ and(X, Y, Z),\ neg(X, Y) \quad \Leftrightarrow \quad neg(X, Y),\ Z=0. \]
Example: Boolean Constraints

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X \wedge Y</th>
<th>X \lor Y</th>
<th>\neg X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Simplification Rules

\begin{align*}
\text{and}(0, Y, Z) & \iff Z=0. \\
\text{and}(X, X, Z) & \iff X=Z. \\
\text{neg}(X, X) & \iff \text{false}. \\
\text{and}(X, Y, Z), \text{neg}(X, Y) & \iff \text{neg}(X, Y), \ Z=0.
\end{align*}

Propagation Rules

\begin{align*}
\text{and}(X, Y, Z), \text{or}(Z, Y, W) & \implies Y=W. \\
\text{and}(X, Y, Z), \text{or}(X, W, Z) & \implies Z=X.
\end{align*}
Step 1: Generation of propagation rules

[Abdennadher and Rigotti, CP’00 and CP’01]

\[(0;Y;Z) \Rightarrow Z = 0\]

and

\[(X;Y;Z) \lor (Z;Y;Z_1) \Rightarrow Y = Z_1\]

and

\[(X;Y;Z) \land \neg(X;Y) \Rightarrow Z = 0\]

Step 2: Transformation of propagation rules into simplification rules

[Abdennadher and Rigotti, PPDP’01]

\[(0;Y;Z) \Rightarrow Z = 0\]

and

\[(X;Y;Z) \lor (Z;Y;Z_1) \Rightarrow Y = Z_1\]

and

\[(X;Y;Z) \land \neg(X;Y) \land \neg(X;Y) \Rightarrow Z = 0\]
Step 1: Generation of propagation rules

[Abdennadher and Rigotti, CP’00 and CP’01]

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

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

\[
\text{and}(X, Y, Z), \text{neg}(X, Y) \Rightarrow Z = 0.
\]
Step 1: Generation of propagation rules

[Abdennadher and Rigotti, CP’00 and CP’01]

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

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

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

Step 2: Transformation of propagation rules into simplification rules

[Abdennadher and Rigotti, PPDP’01]
Step 1: Generation of propagation rules
[Abdennadher and Rigotti, CP’00 and CP’01]

\[\text{and}(0, Y, Z) \Rightarrow Z=0.\]
\[\text{and}(X, Y, Z), \ or(Z, Y, Z_1) \Rightarrow Y=Z_1.\]
\[\text{and}(X, Y, Z), \ neg(X, Y) \Rightarrow Z=0.\]

Step 2: Transformation of propagation rules into simplification rules
[Abdennadher and Rigotti, PPDP’01]

\[\text{and}(0, Y, Z) \iff Z=0.\]
Step 1: Generation of propagation rules

[Abdennadher and Rigotti, CP’00 and CP’01]

\[
\begin{align*}
\text{and}(0, Y, Z) & \Rightarrow Z=0. \\
\text{and}(X, Y, Z), \text{or}(Z, Y, Z1) & \Rightarrow Y=Z1. \\
\text{and}(X, Y, Z), \text{neg}(X, Y) & \Rightarrow Z=0.
\end{align*}
\]

Step 2: Transformation of propagation rules into simplification rules

[Abdennadher and Rigotti, PPDP’01]

\[
\begin{align*}
\text{and}(0, Y, Z) & \Leftrightarrow Z=0. \\
\text{and}(X, Y, Z), \text{or}(Z, Y, Z1) & \Rightarrow Y=Z1.
\end{align*}
\]
Step 1: Generation of propagation rules

[Abdennadher and Rigotti, CP’00 and CP’01]

\[\text{and}(0, Y, Z) \Rightarrow Z=0.\]
\[\text{and}(X, Y, Z), \ or(Z, Y, Z1) \Rightarrow Y=Z1.\]
\[\text{and}(X, Y, Z), \ neg(X, Y) \Rightarrow Z=0.\]

Step 2: Transformation of propagation rules into simplification rules

[Abdennadher and Rigotti, PPDP’01]

\[\text{and}(0, Y, Z) \Leftrightarrow Z=0.\]
\[\text{and}(X, Y, Z), \ or(Z, Y, Z1) \Rightarrow Y=Z1.\]
\[\text{and}(X, Y, Z), \ neg(X, Y) \Leftrightarrow neg(X, Y), \ Z=0.\]
Overview

- Automatic Generation of Constraint Solvers
  - Generation of Propagation Rules
  - Generation of Simplification Rules
- Constraint Handling Rules
  - Program Analysis
  - Applications
  - Implementation
- Conclusions
Syntax

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

where \( C_L \) and \( C_R \) are sets of atomic constraints
**PROPMiner Algorithm**

[Abdennadher and Rigotti, 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**  
[Abdennadher and Rigotti, 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 $C_L \models \bot$, then $C_L \Rightarrow false$

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

if $C_R \neq \emptyset$, then $C_L \Rightarrow C_R$
Example: Boolean Conjunction

\[\begin{array}{c|c|c}
X & Y & X \land Y \\
\hline
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}\]

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

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

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X &amp; 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>

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)
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>

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

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

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

$\text{and}(X, Y, Z), \ X=0$
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>

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

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

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

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

$\text{and}(X, Y, Z), \ X=0, \ Y=0$
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>

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

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

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

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

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

\[
\ldots
\]

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

\[
\ldots
\]
Example: Boolean Conjunction

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X &amp; 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>

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

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

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

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

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

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

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

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

$\ldots$

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

$\ldots$
Example: Boolean Conjunction

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X ∧ 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>

*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.
\ldots
**PROPMiner Algorithm**

[Abdennadher and Rigotti, 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**

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

- if \( C_L \models \bot \), 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 \( C_L \Rightarrow C_R \)
Pruning Strategies

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

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 $C \cap C_R \neq \emptyset$. 
Pruning Strategies

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:

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

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

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:

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

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

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:

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

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

$\text{and}(X, Y, Z), \text{neg}(A, B), \ A=X, \ B=Y, \ B=1$
Pruning Strategies

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:

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

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

$and(X, Y, Z), \ neg(A, B), \ A=X, \ B=Y, \ B=1$ 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
[Abdennadher and Rigotti, CP’00 and PPDP’01]

- **Boolean Constraints**: > 100 rules for \( \neg, \land, \lor, \oplus \)
- **Temporal Reasoning** (Allen’s Interval Approach): 489 rules for composition
- **Spatial Reasoning** (Region Connection Calculus): 178 rules for composition
- **Automatic Test Pattern Generation**: > 2000 rules for six-valued logic
- **Crossword Compilation**:
  
  \[
  \begin{align*}
  w_6(b,e,t,t,e,r) &. \quad w_5(b,r,a,k,e). \quad w_4(b,u,m,p). \\
  w_6(c,a,n,n,o,n) &. \quad w_5(b,l,o,k,e). \quad w_4(p,l,a,y). \\
  w_6(w,e,a,l,t,h) &. \quad w_5(s,t,e,a,m). \quad w_4(f,r,e,e). \\
  w_6(d,e,a,r,t,h) &. \quad w_5(c,r,e,a,m). \quad w_4(s,t,o,p). \\
  w_5(p,a,t,c,h) &. \quad w_5(p,i,t,c,h).
  \end{align*}
  \]
Generation of Propagation Rules VI

Applications

[Abdennadher and Rigotti, CP’00 and PPDP’01]

- **Boolean Constraints**: > 100 rules for \( \neg, \land, \lor, \oplus \)
- **Temporal Reasoning** (Allen’s Interval Approach): 489 rules for composition
- **Spatial Reasoning** (Region Connection Calculus): 178 rules for composition
- **Automatic Test Pattern Generation**: > 2000 rules for six-valued logic
- **Crossword Compilation**:

  - \( w_6(b, e, t, t, e, r) \)
  - \( w_6(c, a, n, n, o, n) \)
  - \( w_6(w, e, a, l, t, h) \)
  - \( w_6(d, e, a, r, t, h) \)
  - \( w_5(b, r, a, k, e) \)
  - \( w_5(b, l, o, k, e) \)
  - \( w_5(s, t, e, a, m) \)
  - \( w_5(c, r, e, a, m) \)
  - \( w_5(p, a, t, c, h) \)
  - \( w_5(p, i, t, c, h) \)

![Crossword Grid](image)
Applications

[Abdennadher and Rigotti, CP’00 and PPDP’01]

- **Boolean Constraints**: > 100 rules for ¬, ∧, ∨, ⊕
- **Temporal Reasoning** (Allen’s Interval Approach): 489 rules for composition
- **Spatial Reasoning** (Region Connection Calculus): 178 rules for composition
- **Automatic Test Pattern Generation**: > 2000 rules for six-valued logic
- **Crossword Compilation**:

  w6(b,e,t,t,e,r).
  w6(c,a,n,n,o,n).
  w6(w,e,a,l,t,h).
  w6(d,e,a,r,t,h).
  w5(b,r,a,k,e).
  w5(b,l,o,k,e).
  w5(s,t,e,a,m).
  w5(d,e,a,r,t,h).
  w5(p,a,t,c,h).
  w5(p,i,t,c,h).

```
 1 2 3 4 5 6
A  s t o p
B  t
C  e
D  a
E  m
```
Applications

[Abdennadher and Rigotti, CP’00 and PPDP’01]

- **Boolean Constraints**: > 100 rules for $\neg, \land, \lor, \oplus$
- **Temporal Reasoning** (Allen’s Interval Approach): 489 rules for composition
- **Spatial Reasoning** (Region Connection Calculus): 178 rules for composition
- **Automatic Test Pattern Generation**: > 2000 rules for six-valued logic
- **Crossword Compilation**:

  w6(b,e,t,t,e,r).
  w6(c,a,n,n,o,n).
  w6(w,e,a,l,t,h).
  w6(d,e,a,r,t,h).
  w5(b,r,a,k,e).
  w5(b,l,o,k,e).
  w5(s,t,e,a,m).
  w5(d,e,a,r,t,h).
  w5(w,e,a,l,t,h).
  w5(c,r,e,a,m).
  w5(p,a,t,c,h).
  w5(p,i,t,c,h).

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
A & s & t & o & p \\
B & t & & & & \\
C & e & a & t & h \\
D & a & & & & \\
E & m & & & & \\
\end{array}
\]
Applications

[Abdennadher and Rigotti, CP’00 and PPDP’01]

- **Boolean Constraints**: > 100 rules for $\neg, \land, \lor, \oplus$
- **Temporal Reasoning** (Allen’s Interval Approach): 489 rules for composition
- **Spatial Reasoning** (Region Connection Calculus): 178 rules for composition
- **Automatic Test Pattern Generation**: > 2000 rules for six-valued logic
- **Crossword Compilation**:

  \[
  \begin{align*}
  w6(b,e,t,t,e,r). & \quad w5(b,r,a,k,e). & \quad w4(b,u,m,p). \\
  w6(c,a,n,n,o,n). & \quad w5(b,l,o,k,e). & \quad w4(p,l,a,y). \\
  w6(w,e,a,l,t,h). & \quad w5(s,t,e,a,m). & \quad w4(f,r,e,e). \\
  w6(d,e,a,r,t,h). & \quad w5(c,r,e,a,m). & \quad w4(s,t,o,p). \\
  w5(p,a,t,c,h). & \quad w5(p,i,t,c,h). & \\
  \end{align*}
  \]
Overview

- Automatic Generation of Constraint Solvers
  - Generation of Propagation Rules
  - Generation of Simplification Rules
- Constraint Handling Rules
  - Program Analysis
  - Applications
  - Implementation
- Conclusions
Rule-based Constraint Solvers

Example: Boolean Constraints

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X ∧ Y</th>
<th>X ∨ Y</th>
<th>¬X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Simplification Rules

\[
\begin{align*}
\text{and}(0, Y, Z) & \iff Z = 0. \\
\text{and}(X, X, Z) & \iff X = Z. \\
\text{and}(X, Y, 1) & \iff X = 1, \ Y = 1. \\
\text{neg}(X, X) & \iff \text{false}.
\end{align*}
\]

Propagation Rules

\[
\begin{align*}
\text{and}(X, Y, Z), \ \text{or}(Z, Y, W) & \Rightarrow Y = W. \\
\text{and}(X, Y, Z), \ \text{or}(X, W, Z) & \Rightarrow Z = X.
\end{align*}
\]
Motivation

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

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

Removing constraints

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

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

Removing constraints

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

**Aim:** Find criteria to transform some propagation rules into simplification rules
Syntactical Criterion: Confluence

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

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

1. \( \text{and}(X, Y, Z), \text{neg}(X, Y) \Leftrightarrow Z=0 \)
Generation of Simplification Rules II

Syntactical Criterion: Confluence

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

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

2. \( \text{and}(X, Y, Z), \text{neg}(X, Y) \Leftrightarrow \text{and}(X, Y, Z), Z=0 \)
Syntactical Criterion: Confluence

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

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

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

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

- Automatic Generation of Constraint Solvers
  - Generation of Propagation Rules
  - Generation of Simplification Rules
- **Constraint Handling Rules**
  - Program Analysis
  - Applications
  - Implementation
- Conclusions
Declarative programming language for the specification and implementation of constraint solvers and programs.

Host language (Prolog, Haskell, Java,...)
**Consistency**
The logical meaning of the rules is consistent.
[Abdennadher et al., Constraints Journal 2000]

**Confluence**
The answer of a query is always the same, no matter which of the applicable rules are applied.
[Abdennadher et al., CP’96, CP’97, Constraints Journal 2000]

**Completion**
Make non-confluent solvers confluent by adding new rules.
[Abdennadher and Frühwirth, CP’98]

**Operational Equivalence**
Do two programs have the same behavior?
[Abdennadher and Frühwirth, CP’99]
CHR Applications

Munich Rent Advisor
[Frühwirth and Abdennadher, TPLP’01]

University Timetabling
[Abdennadher and Marte, AAI’00]

Nurse Scheduling
[Abdennadher and Schlenker, IAAI’99]

Classroom Assignment
[Abdennadher, Saft and Will, PACLP’00]
CHR Implementations

- Eclipse Prolog, YAP Prolog, Sicstus Prolog (CHR online)
  
  www.pms.informatik.uni-muenchen.de/~webchr/

- Haskell

- JACK (Java Constraint Kit) [Abdennadher et al., WFLP’01 and WLPE’01]
  - JCHR: Java Constraint Handling Rules
  - JASE: Java Abstract Search Engine
  - VisualCHR: An interactive tool to visualize JCHR computations
Conclusions

**Constraint Programming**

Generic Framework for

- modelling with incomplete information
- solving of combinatorial problems

**Constraint Handling Rules**

Declarative language for constraint programming

- Executable specification and rapid prototyping
- Good theoretical properties
- Implementation and libraries available
- Automatic Generation
Future Work

- Automatic generation of constraint solvers
  - Generation of solvers for intentionally defined constraints:

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

    \[
    \min(A, B, C) \ \Downarrow \ \Rightarrow \ C \leq A, \ C \leq B.
    \]

    \[
    \min(A, A, C) \ \Leftrightarrow \ A = C.
    \]

    \[
    \begin{align*}
    \min(A, B, C), \ C \not\equiv B & \Rightarrow \ C = A. \\
    \min(A, B, C), \ C \not\equiv A & \Rightarrow \ C = B. \\
    \min(A, B, C), \ B \leq A & \Rightarrow \ C = B, \ B \leq A. \\
    \min(A, B, C), \ A \leq B & \Rightarrow \ C = A, \ A \leq B.
    \end{align*}
    \]

  - Further applications: e.g. Security Policies

- Semantic Web and Constraint Programming
  - Intelligent Agents: e.g. Constrained Shopping Cart, Appointment Scheduling
  - Ontologies and Constraint Programming
Projects

- **JACK**: JAva Constraint Kit, project leader since 2002, Project partners are FAST Munich, Siemens AG Munich, Instituto de Sistemas, University Tandil, Argentina (IB-BMBF/SCyT Project ARG 030/98 INF).
- **Extraction de Connaissances pour la Construction Automatisée de Solveurs de Contraintes Efficaces**: Projet de coopération entre l’Université de Munich et l’INSA de Lyon (Centre de Coopération Universitaire Franco-Bavarois)
- **Automatic Constraint Solving: Theory and Practice**: DFG Project (to be submitted)
**SIMPMiner Algorithm**

**INPUT:** A terminating program $P$ consisting of propagation rules

**OUTPUT:** A program $P'$ 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 $(P'\{R\}) \cup \{R'\}$ is terminating and confluent.

**If** $R'$ exists

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

Automatic test-pattern generation

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

- Single-headed propagation rules (77 rules)
- Propagation rules with one or two atoms in the head (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%