ASP modulo CSP: The clingcon system

Max Ostrowski and Torsten Schaub

University of Potsdam
1. The clingcon System
2. Learning
3. Benchmarks
The \textit{clingcon} System

- ASPmCSP solver
- ASP Answer Set Programming + CP solver
- SMT style (SAT + Theory)
  - lazy (no translation)
  - incremental (check of partial assignments)
  - online (backjumping and learning)
  - theory propagation
ASP vs. SAT

- Input Language with First Order Variables

"from a syntactic point of view the language is ugly, would be a torture to use, and is nearly impossible to read"
ASP vs. SAT

- Input Language with First Order Variables
  - “from a syntactic point of view the language is ugly, would be a torture to use, and is nearly impossible to read”
N-Queens

% place n queens on the chess board
n [ q(1..n,1..n) ] n.
% at most one queen per row/column
   :- q(X,Y1), q(X,Y2), Y1 < Y2.
   :- q(X1,Y), q(X2,Y), X1 < X2.
% at most one queen per diagonal
   :- q(X1,Y1), q(X2,Y2),
      #abs(X1 - X2) == #abs(Y1 - Y2),
      X1 < X2, Y1 != Y2.
% place n queens on the chess board
n [ q(1..n,1..n) ] n.
% at most one queen per row/column
:- q(X,Y1), q(X,Y2), Y1 < Y2.
:- q(X1,Y), q(X2,Y), X1 < X2.
% at most one queen per diagonal
:- q(X1,Y1), q(X2,Y2),
   #abs(X1 - X2) == #abs(Y1 - Y2),
   X1 < X2, Y1 != Y2.
% place n queens on the chess board
n [ q(1..n,1..n) ] n.
% at most one queen per row/column
   :- q(X,Y1), q(X,Y2), Y1 < Y2.
   :- q(X1,Y), q(X2,Y), X1 < X2.
% at most one queen per diagonal
   :- q(X1,Y1), q(X2,Y2),
      #abs(X1 - X2) == #abs(Y1 - Y2),
      X1 < X2, Y1 != Y2.
% place n queens on the chess board
n [ q(1..n,1..n) ] n.
% at most one queen per row/column
:- q(X,Y1), q(X,Y2), Y1 < Y2.
:- q(X1,Y), q(X2,Y), X1 < X2.
% at most one queen per diagonal
:- q(X1,Y1), q(X2,Y2),
    #abs(X1 - X2) == #abs(Y1 - Y2),
    X1 < X2, Y1 != Y2.
N-Queens

% place n queens on the chess board
n [ q(1..n,1..n) ] n.

% at most one queen per row/column
  :- q(X,Y1), q(X,Y2), Y1 < Y2.
  :- q(X1,Y), q(X2,Y), X1 < X2.

% at most one queen per diagonal
  :- q(X1,Y1), q(X2,Y2),
      #abs(X1 - X2) == #abs(Y1 - Y2),
      X1 < X2, Y1 != Y2.

Declarative!
% Select edges for the cycle
1 \{ cycle(X,Y) : edge(X,Y), cycle(X,Y) : edge(Y,X) \} 1 :- vtx(X).
1 \{ cycle(X,Y) : edge(X,Y), cycle(X,Y) : edge(Y,X) \} 1 :- vtx(Y).

reached(X) :- bound(X).
reached(Y) :- reached(X), cycle(X,Y).

:- vtx(X), not reached(X).
TSP

% Select edges for the cycle
1 { cycle(X,Y) : edge(X,Y), cycle(X,Y) : edge(Y,X) } 1 :- vtx(X).
1 { cycle(X,Y) : edge(X,Y), cycle(X,Y) : edge(Y,X) } 1 :- vtx(Y).

reached(X) :- bound(X).
reached(Y) :- reached(X), cycle(X,Y).

:- vtx(X), not reached(X).

#minimize [ cycle(X,Y) : cost(X,Y,C) = C ].
ASP vs. SAT

- Input Language with First Order Variables
- ASP allows for solving all search problems in \( NP \) (and \( \text{NP}^{\text{NP}} \)) in a uniform way (being more compact than SAT)
- \textit{clasp} based on CDCL (SAT 2011 Competition 1st Crafted UNSAT)
- inbuilt reachability check

Consider the logical formula \( \Phi \) and its three (classical) models:
\[
\{ p, q \}, \quad \{ q, r \}, \quad \text{and} \quad \{ p, q, r \}.
\]

This formula has one answer set:
\[
\{ p, q \}.
\]
ASP vs. SAT

- Input Language with First Order Variables
- ASP allows for solving all search problems in $NP$ (and $NP^{NP}$) in a uniform way (being more compact than SAT)
- clasp based on CDCL (SAT 2011 Competition 1st Crafted UNSAT)
- inbuilt reachability check

Consider the logical formula $\Phi$ and its three (classical) models:

$$\{p, q\}, \{q, r\}, \text{ and } \{p, q, r\}.$$  

This formula has one answer set:

$$\{p, q\}.$$  

$$\Phi = q \land (q \land \lnot r \rightarrow p)$$  

$$\Pi = \begin{align*}  q & \leftarrow \\  p & \leftarrow q, \text{ not } r \end{align*}$$
The *clingcon* Language

- Constraints over integers can be seen as atoms (as in SMT)
- \( x + y > z - 3 \) is either true or false
The \textit{clingcon} Language

- Constraints over integers can be seen as atoms (as in SMT).
- $x + y > z - 3$ is either true or false.

\[
\begin{align*}
x + y & > z : -a, not \ b. \\
a & : -x \succ y, not \ b.
\end{align*}
\]
The *clingcon* Language

- constraints over integers can be seen as atoms (as in SMT)
- \(x + y > z - 3\) is either true or false

\[
x \geq 1 \quad y < 5\]

- global constraints

\[
\text{distinct}\{\text{val}(X) : \text{b}(X) : \text{not d}(X)\}.
\]
The **clingcon** Language

- constraints over integers can be seen as atoms (as in SMT)
- $x + y > z - 3$ is either true or false

$x \mathbin{\!+\!} y \mathbin{\!>\!} z : -a, not\ b.$

\[
a : -x \mathbin{\!>\!} y, not\ b.
\]

- global constraints

$\$distinct\{val(X) : b(X) : not\ d(X)\}.$

- optimize statements

$\$minimize\{cost(X, Y) : edge(X, Y)\}.$
Algorithm 1: CDCL-ASP$\text{MCSP}$

**Input**: A program $\Pi$.

**Output**: A constraint answer set of $\Pi$.

1. loop
   2. Propagation
   3. if `hasConflict` then
      4. if `decisionLevel = 0` then return no Answer Set
      5. `ConflictAnalysis`
      6. `Backjump`
   7. else if `complete Assignment` then
      8. `Labeling`
      9. if `hasConflict` then
         10. `Backjump`
      11. else
         12. return Constraint Answer Set
   13. else
      14. `Select`
Propagation

- Unit Propagation
- Unfounded Set Check
- Constraint Propagation
Propagation

- Unit Propagation
- Unfounded Set Check
- Constraint Propagation
  - use of reified constraints
  - propagate the truthvalue of all yet decided constraints
Propagation

- Unit Propagation
- Unfounded Set Check
- Constraint Propagation
  - use of reified constraints
  - propagate the truthvalue of all yet decided constraints
  - 1. a new constraint can be derived (true or false)
Propagation

- Unit Propagation
- Unfounded Set Check
- Constraint Propagation
  - use of reified constraints
  - propagate the truthvalue of all yet decided constraints
  - 1. a new constraint can be derived (true or false)
  - 2. domain of variable became empty (conflict)
Conflict

- no clue what caused the conflict
- just take all information (all yet decided constraints)
Conflict

- no clue what caused the conflict
- just take all information (all yet decided constraints)
  - usually very large
  - quite unspecific
- minimizing this inconsistent set to an IIS
- QuickXPlain (Junker’01)
IIS: No constraint can be removed
Algorithm 2: DELETION_FILTERING

**Input**: An inconsistent list of constraints $I = [c_1, \ldots, c_n]$.

**Output**: An irreducible inconsistent list of constraints.

1. $i \leftarrow 1$
2. **while** $i \leq |I|$ **do**
3.  **if** $I \setminus c_i$ is inconsistent **then**
4.  $I \leftarrow I \setminus c_i$
5.  **else**
6.  $i \leftarrow i + 1$
7. **return** $I$
Deletion Filtering - Example

\[ I = [\text{work}(\text{lea}) = \text{work}(\text{adam}), \text{work}(\text{john}) = 0, \text{work}(\text{smith}) = 0] \]

\[ \circ [\text{work}(\text{adam}) + \text{work}(\text{lea}) > 6, \text{work}(\text{lea}) - \text{work}(\text{adam}) = 1] \]
Deletion Filtering - Example

\[ l = [\text{work}(\text{lea}) = \text{work}(\text{adam}), \text{work}(\text{john}) = 0, \text{work}(\text{smith}) = 0] \]
\[ \circ [\text{work}(\text{adam}) + \text{work}(\text{lea}) > 6, \text{work}(\text{lea}) - \text{work}(\text{adam}) = 1] \]
Deletion Filtering - Example

\[ l = [\text{work}\,(\text{lea}) = \text{work}\,(\text{adam}), \text{work}\,(\text{john}) = 0, \text{work}\,(\text{smith}) = 0] \]

\[ \circ [\text{work}\,(\text{adam}) + \text{work}\,(\text{lea}) > 6, \text{work}\,(\text{lea}) - \text{work}\,(\text{adam}) = 1] \]
Deletion Filtering - Example

\[ l = [work(\text{lea}) = work(\text{adam}),
\quad \circ \ [work(\text{adam}) + work(\text{lea}) > 6, work(\text{lea}) - work(\text{adam}) = 1] \]
Deletion Filtering - Example

\[ l = [\text{work} (\text{lea}) = \text{work} (\text{adam})], \]
\[ \text{o } [\text{work} (\text{adam}) + \text{work} (\text{lea}) \geq 6, \text{work} (\text{lea}) - \text{work} (\text{adam}) = 1] \]
Deletion Filtering - Example

\[ l = [\text{work}(\text{lea}) = \text{work}(\text{adam})], \]
\[ \circ [\text{work}(\text{lea}) - \text{work}(\text{adam}) = 1] \]
Deletion Filtering - Example

\[
I = \left[ \begin{array} { c c }
\text{work(lea)} = \text{work(adam)}, \\
\text{work(lea)} - \text{work(adam)} = 1
\end{array} \right]
\]
Algorithm 3: FORWARD_FILTERING

Input: An inconsistent list of constraints $I = [c_1, \ldots, c_n]$.

Output: An irreducible inconsistent list of constraints $I'$.

1. $I' \leftarrow []$

2. while $I'$ is consistent do

3. \hspace{1em} $T \leftarrow I'$

4. \hspace{1em} $i \leftarrow 1$

5. \hspace{2em} while $T$ is consistent do

6. \hspace{3em} $T \leftarrow T \circ c_i$

7. \hspace{3em} $i \leftarrow i + 1$

8. \hspace{2em} $I' \leftarrow I' \circ c_i$

9. return $I'$
Forward Filtering - Example

\[
I = \begin{bmatrix}
\text{work (adam)} & \\
\circ & \\
\text{work (john)} & \\
\circ & \\
\text{work (smith)} & \\
\circ & \\
\end{bmatrix}
\]
Forward Filtering - Example

\[ I = \left[ \begin{array}{c}
work(\text{lea}) = work(\text{adam}), \\
\circ [ \\
\end{array} \right] \]

\[ \begin{array}{c}
\circ [ \\
\end{array} \right] \]
\[ I = [\text{work}(lea) = \text{work}(adam), \text{work}(john) = 0, \text{work}(smith) = 0] \]

\[ \circ [\quad , \quad ] \]
Forward Filtering - Example

\[ l = [\text{work}(\text{lea}) = \text{work}(\text{adam}), \text{work}(\text{john}) = 0, \text{work}(\text{smith}) = 0] \]

\[ \circ [\text{work}(\text{adam}) + \text{work}(\text{lea}) > 6, ] \]
$l = [\text{work}(\text{lea}) = \text{work}(\text{adam}), \text{work}(\text{john}) = 0, \text{work}(\text{smith}) = 0]$

$\circ [\text{work}(\text{adam}) + \text{work}(\text{lea}) > 6, \text{work}(\text{lea}) - \text{work}(\text{adam}) = 1]$
Forward Filtering - Example

\[ I = [work(\text{lea}) = work(\text{adam}), work(\text{john}) = 0, work(\text{smith}) = 0] \]

\[ \circ [work(\text{adam}) + work(\text{lea}) > 6, work(\text{lea}) - work(\text{adam}) = 1] \]
Forward Filtering - Example

\[ I = [work(adam), work(lea) > 6], work(lea) - work(adam) = 1] \]
Forward Filtering - Example

\[
I = [\text{work}(\text{lea}) = \text{work}(\text{adam})],
\]

\[
\circ [\text{work}(\text{lea}) - \text{work}(\text{adam}) = 1]
\]
Derivations

- Forward
- Backward
- ConnectedComponent
- Range
- ConnectedComponentRange
Whenever we do theory propagation, we need a reason

- simplest reason is again all yet decided constraints
- minimize reason set
Reasons

Whenever we do theory propagation, we need a reason

- simplest reason is again all yet decided constraints
- minimize reason set
- every reason can be seen as an IIS
- \{work(john) = 0, work(lea) - work(adam) = 1\} is the reason for work(lea) \neq work(adam)
Whenever we do theory propagation, we need a reason

- simplest reason is again all yet decided constraints
- minimize reason set
- every reason can be seen as an IIS

\[ \{ \text{work}(john) = 0, \text{work}(lea) - \text{work}(adam) = 1 \} \] is the reason for \( \text{work}(lea) \neq \text{work}(adam) \)

\[ \{ \text{work}(john) = 0, \text{work}(lea) - \text{work}(adam) = 1, \text{work}(lea) = \text{work}(adam) \} \]
Benchmarks

- Packing
- Incremental Scheduling
- Weighted Assignment Tree (join-order optimization of SQL)
- Quasi Group
- Unfounded Set Check
Benchmarks - Average Conflict Size

(a) Packing

(b) Inc. Shed

(c) Quasi Group

(d) Weighted Tree

(e) USC
Benchmarks - Average Time

(a) Packing

(b) Inc. Sched.

(c) Quasi Group

(d) Weighted Tree

(e) USC
## Benchmarks

<table>
<thead>
<tr>
<th>Instances</th>
<th>time</th>
<th>time</th>
<th>acs</th>
<th>acs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s/s</td>
<td>o/b</td>
<td>s/s</td>
<td>o/b</td>
</tr>
<tr>
<td><strong>Packing</strong> (50)</td>
<td>888 (49)</td>
<td>63 (0)</td>
<td>293</td>
<td>40</td>
</tr>
<tr>
<td><strong>Inc. Sched.</strong> (50)</td>
<td>30 (01)</td>
<td>3 (0)</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td><strong>Quasi Group</strong> (78)</td>
<td>390 (28)</td>
<td>12 (0)</td>
<td>480</td>
<td>56</td>
</tr>
<tr>
<td><strong>Weighted Tree</strong> (30)</td>
<td>484 (07)</td>
<td>574 (18)</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td><strong>USC</strong> (132)</td>
<td>721 (104)</td>
<td>92 (1)</td>
<td>454</td>
<td>13</td>
</tr>
</tbody>
</table>
Questions

Outlook!

Looking for expert knowledge about SMT systems