ASP modulo CSP: The clingcon system

Max Ostrowski and Torsten Schaub

University of Potsdam

Outline

The *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

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.
```

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.
```

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.
```

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.
```

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!

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).
```

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 NP^{NP}) in a uniform way (being more compact than SAT)
- *clasp* based on CDCL (SAT 2011 Competition 1st Crafted UNSAT)
- inbuilt reachability check

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 Φ and its three (classical) models:

$$\Phi \quad \boxed{q \ \wedge \ (q \wedge \neg r \rightarrow p)}$$

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

This formula has one answer set:

$$\Pi \quad \boxed{\begin{array}{lll} q & \leftarrow & \\ p & \leftarrow & q, \ not \ r \end{array}}$$

$$\{p, q\}$$

The *clingcon* Language

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

The *clingcon* Language

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

$$\begin{array}{rl} x \ \$+ \ y \ \$> \ z & :- a, not \ b. \\ a & :- x \ \$> \ y, not \ b. \end{array}$$

The *clingcon* Language

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

$$\begin{array}{rl} x \ \$\!+\ y \ \$\!>\ z & :- a, not\ \ b. \\ a & :- x \ \$\!>\ y, not\ \ b. \end{array}$$

- global constraints

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

The *clingcon* Language

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

$$\begin{aligned} x \ \$+\ y \ \$>\ z \quad &:- a, not\ b. \\ a \quad &:- x \ \$>\ y, not\ b. \end{aligned}$$

- global constraints

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

- optimize statements

$$\$minimize\{cost(X, Y) : edge(X, Y)\}.$$

CDCL

Algorithm 1: CDCL-ASPmCSP

Input : A program Π.
Output : A constraint answer set of Π.

```
loop
    Propagation
    if hasConflict then
        if decisionLevel = 0 then return no Answer Set
        ConflictAnalysis
        Backjump
    else if complete Assignment then
        Labeling
        if hasConflict then
            Backjump
        else
            return Constraint Answer Set
    else
        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

IIS: No constraint can be removed

IIS

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.

$i \leftarrow 1$
while $i \leq |I|$ **do**
  if $I \setminus c_i$ *is inconsistent* **then**
    $I \leftarrow I \setminus c_i$
  else
    $i \leftarrow i + 1$
return I

Deletion Filtering - Example

$$I = [work(lea) = work(adam), work(john) = 0, work(smith) = 0]$$
$$\circ\ [work(adam) + work(lea) > 6, work(lea) - work(adam) = 1]$$

Deletion Filtering - Example

$$I = [\sout{work(lea) = work(adam)}, work(john) = 0, work(smith) = 0]$$
$$\circ\ [work(adam) + work(lea) > 6, work(lea) - work(adam) = 1]$$

Deletion Filtering - Example

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

$\circ\ [\mathit{work}(\mathit{adam}) + \mathit{work}(\mathit{lea}) > 6, \mathit{work}(\mathit{lea}) - \mathit{work}(\mathit{adam}) = 1]$

Deletion Filtering - Example

$$I = [work(lea) = work(adam), \qquad]$$

$$\circ\ [work(adam) + work(lea) > 6, work(lea) - work(adam) = 1]$$

Deletion Filtering - Example

$I = [work(lea) = work(adam), \quad]$

$\circ\ [\sout{work(adam) + work(lea) > 6}, work(lea) - work(adam) = 1]$

Deletion Filtering - Example

$$I = [work(lea) = work(adam), \qquad]$$

$$\circ\ [\qquad , \sout{work(lea) - work(adam) = 1}]$$

Deletion Filtering - Example

$$I = [work(lea) = work(adam), \quad]$$
$$\circ\ [\quad , work(lea) - work(adam) = 1]$$

IIS

IIS: No constraint can be removed

Algorithm 3: FORWARD_FILTERING

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

```
I' ← []
while I' is consistent do
    T ← I'
    i ← 1
    while T is consistent do
        T ← T ∘ c_i
        i ← i + 1
    I' ← I' ∘ c_i
return I'
```

Forward Filtering - Example

$$l = [\quad , \quad]$$
$$\circ [\quad , \quad]$$

Forward Filtering - Example

$$I = [work(lea) = work(adam), \qquad]$$
$$\circ\ [\qquad , \qquad]$$

Forward Filtering - Example

$$I = [work(lea) = work(adam), work(john) = 0, work(smith) = 0]$$
$$\circ\ [\qquad , \qquad]$$

Forward Filtering - Example

$$I = [work(lea) = work(adam), work(john) = 0, work(smith) = 0]$$
$$\circ\ [work(adam) + work(lea) > 6, \qquad]$$

Forward Filtering - Example

$$I = [work(lea) = work(adam), work(john) = 0, work(smith) = 0]$$
$$\circ\ [work(adam) + work(lea) > 6, work(lea) - work(adam) = 1]$$

Forward Filtering - Example

$$I = [work(lea) = work(adam), work(john) = 0, work(smith) = 0]$$
$$\circ\ [work(adam) + work(lea) > 6, work(lea) - work(adam) = 1]$$

Forward Filtering - Example

$$I = [\qquad\qquad\qquad\qquad , \qquad\qquad\qquad\qquad]$$

$$\circ\ [\qquad\qquad\qquad\qquad , work(lea) - work(adam) = 1]$$

Forward Filtering - Example

$$I = [work(lea) = work(adam), \qquad\qquad\qquad\qquad\qquad\qquad]$$
$$\circ\ [\qquad\qquad\qquad\qquad\qquad\qquad , work(lea) - work(adam) = 1]$$

Derivations

- *Forward*
- *Backward*
- *ConnectedComponent*
- *Range*
- *ConnectedComponentRange*

Reasons

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 $work(lea) \neq work(adam)$

Reasons

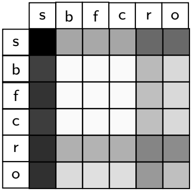
Whenever we do theory propagation, we need a reason

- 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 $work(lea) \neq work(adam)$
- $\{work(john) = 0, work(lea) - work(adam) = 1, work(lea) = work(adam)\}$

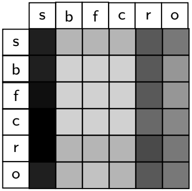
Benchmarks

- Packing
- Incremental Sheduling
- Weighted Assignment Tree (join-order optimization of SQL)
- Quasi Group
- Unfounded Set Check

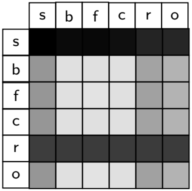
Benchmarks - Average Conflict Size



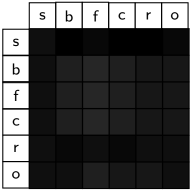
(a) Packing



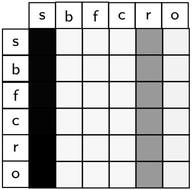
(b) Inc. Shed



(c) Quasi Group

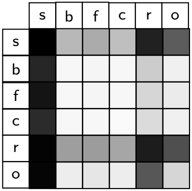


(d) Weighted Tree

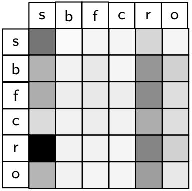


(e) USC

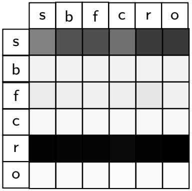
Benchmarks - Average Time



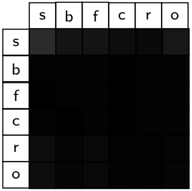
(a) Packing



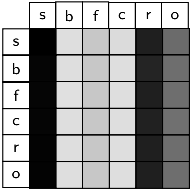
(b) Inc. Sched.



(c) Quasi Group



(d) Weighted Tree



(e) USC

Benchmarks

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

Outlook!

Looking for expert knowledge about SMT systems