

Translation-based Constraint Answer Set Solving

Christian Drescher Toby Walsh

NICTA and University of New South Wales

IJCAI, 20 July 2011



Australian Government
Department of Broadband, Communications
and the Digital Economy
Australian Research Council



Industry &
Investment



Answer Set Programming (ASP)

Syntax and semantics

Answer Set Programming (ASP)

Syntax and semantics

Logic Prog. set P of **rules** over a set of atomic propositions

$$a_1 \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n$$

Answer Set set of atoms X , and least model of the GL-reduct

$$P^X = \{a_1 \leftarrow b_1, \dots, b_m \mid r \in P, c_1, \dots, c_n \cap X = \emptyset\}$$

Example

$$P = \left\{ \begin{array}{ll} \text{swim} & \leftarrow \text{sunny}, \text{not shark} \\ \text{sunny} & \leftarrow \end{array} \right\}$$

X		
$\{\text{sunny}, \text{swim}\}$		
$\{\text{sunny}, \text{shark}\}$		
$\{\text{sunny}, \text{swim}, \text{shark}\}$		

Answer Set Programming (ASP)

Syntax and semantics

Logic Prog. set P of **rules** over a set of atomic propositions

$$a_1 \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n$$

Answer Set set of atoms X , and least model of the GL-reduct

$$P^X = \{a_1 \leftarrow b_1, \dots, b_m \mid r \in P, c_1, \dots, c_n \cap X = \emptyset\}$$

Example

$$P = \left\{ \begin{array}{l} \text{swim} \leftarrow \text{sunny}, \text{not shark} \\ \text{sunny} \leftarrow \end{array} \right\}$$

X	P^X	least model of P^X
$\{\text{sunny}, \text{swim}\}$	$\text{swim} \leftarrow \text{sunny}$ $\text{sunny} \leftarrow$	$\{\text{sunny}, \text{swim}\}$
$\{\text{sunny}, \text{shark}\}$		
$\{\text{sunny}, \text{swim}, \text{shark}\}$		

Answer Set Programming (ASP)

Syntax and semantics

Logic Prog. set P of **rules** over a set of atomic propositions

$$a_1 \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n$$

Answer Set set of atoms X , and least model of the GL-reduct

$$P^X = \{a_1 \leftarrow b_1, \dots, b_m \mid r \in P, c_1, \dots, c_n \cap X = \emptyset\}$$

Example

$$P = \left\{ \begin{array}{l} \text{swim} \leftarrow \text{sunny}, \text{not shark} \\ \text{sunny} \leftarrow \end{array} \right\}$$

X	P^X	least model of P^X
$\{\text{sunny}, \text{swim}\}$	$\text{swim} \leftarrow \text{sunny}$ $\text{sunny} \leftarrow$	$\{\text{sunny}, \text{swim}\}$
$\{\text{sunny}, \text{shark}\}$	$\text{sunny} \leftarrow$	$\{\text{sunny}\}$
$\{\text{sunny}, \text{swim}, \text{shark}\}$	$\text{sunny} \leftarrow$	$\{\text{sunny}\}$

Answer Set Programming (ASP)

Syntax and semantics

Logic Prog. set P of **rules** over a set of atomic propositions

$$a_1 \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n$$

Answer Set set of atoms X , and least model of the GL-reduct

$$P^X = \{a_1 \leftarrow b_1, \dots, b_m \mid r \in P, c_1, \dots, c_n \cap X = \emptyset\}$$

Example

$$P = \left\{ \begin{array}{l} \text{swim} \leftarrow \text{sunny}, \text{not shark} \\ \text{sunny} \leftarrow \end{array} \right\}$$

X	P^X	least model of P^X
$\{\text{sunny}, \text{swim}\}$	$\text{swim} \leftarrow \text{sunny}$ $\text{sunny} \leftarrow$	$\{\text{sunny}, \text{swim}\}$
$\{\text{sunny}, \text{shark}\}$	$\text{sunny} \leftarrow$	$\{\text{sunny}\}$
$\{\text{sunny}, \text{swim}, \text{shark}\}$	$\text{sunny} \leftarrow$	$\{\text{sunny}\}$

Algorithms Conflict-Driven Nogood Learning, Unit-Propagation, Heuristic Search, Conflict Resolution, ...

Constraint Answer Set Programming (CASP)

Syntax and semantics

- Constraint** relation on variables with finite domains
- ~ **Logic Prog.** set P of rules over a set of atomic propositions
and constraints
- ~ **Answer Set** see for instance [Gebser et. al, 2009],
seamlessly extends definition of (regular) answer set

Example

$$P = \left\{ \begin{array}{ll} \textit{swim} & \leftarrow \textit{sunny}, \textit{not shark} \\ \textit{sunny} & \leftarrow \\ \perp & \leftarrow \$\textit{deadline} - \$\textit{day_off} < 5d \\ & \dots \end{array} \right\}$$

Constraint Answer Set Programming (CASP)

Syntax and semantics



- Constraint** relation on variables with finite domains
- ~ **Logic Prog.** set P of rules over a set of atomic propositions and constraints
- ~ **Answer Set** see for instance [Gebser et. al, 2009], seamlessly extends definition of (regular) answer set

Example

$$P = \left\{ \begin{array}{ll} \textit{swim} & \leftarrow \textit{sunny}, \textit{not danger} \\ \textit{sunny} & \leftarrow \\ \perp & \leftarrow \$\textit{deadline} - \$\textit{day_off} < 5d \\ \textit{danger} & \leftarrow \textit{shark} \\ & \dots \\ \textit{danger} & \leftarrow \textit{not } \#all\textit{different}\{\$color_1, \dots, \$color_n\} \end{array} \right\}$$

Towards Constraint Answer Set Solving

Some Related Work

Challenge

Enhancing Answer Set Programming (ASP) by Constraint Programming (CP) techniques, towards

- improved modelling convenience, and
- improved solving speed.

$$\text{CASP} = \text{ASP} + \text{CP}$$

Towards Constraint Answer Set Solving

Some Related Work

Challenge

Enhancing Answer Set Programming (ASP) by Constraint Programming (CP) techniques, towards

- improved modelling convenience, and
- improved solving speed.

$$\text{CASP} = \text{ASP} + \text{CP}$$

[Baselice et al., 2005; Mellarkod et al., 2008]

- Hybrid of CP system and ASP solver on non-ground level

[Gebser et al., 2009]

- SMT-like approach

[Balduccini, 2010]

- Answer sets specify constraint problem

Translation-based Constraint Answer Set Solving

Outline

[Drescher and Walsh, 2010]

- translation-based approach rather than a hybrid one
- efficient propagation and learning for free

Logic Prog. set P of **rules** over a set of atomic propositions

$$a_1 \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n$$

and extensions like **choice rules** of the form

$$\{a_1, \dots, a_k\} \leftarrow b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n$$

and **cardinality rules**

$$a_1 \leftarrow k\{b_1, \dots, b_m, \text{not } c_1, \dots, \text{not } c_n\}$$

- 1 Direct Encoding
- 2 Support Encoding
- 3 Range Encoding
- 4 Bound Encoding

Idea: Replace each constraint c by a regular atom $\text{sat}(c)$, and add rules

$$\begin{aligned} \text{sat}(c) &\leftarrow \text{not } \text{violate}(c) \\ \text{violate}(c) &\leftarrow \text{not } \text{sat}(c) \end{aligned}$$

Direct Encoding

- folklore, similar to [Walsh, 2000]
- new atoms $e(v, i)$, v is assigned the value $i \in \{1, \dots, d\}$, $v = i$
 $\{e(v, 1), \dots, e(v, d)\} \leftarrow$
 $\perp \leftarrow \text{not } e(v, 1), \dots, \text{not } e(v, d)$
 $\perp \leftarrow 2 \{e(v, 1), \dots, e(v, d)\}$
- encode forbidden combination of values $v_1 = i_1, v_2 = i_2, \dots, v_n = i_n$:
 $\text{violate}(c) \leftarrow e(v_1, i_1), e(v_2, i_2), \dots, e(v_n, i_n)$

Direct Encoding

- folklore, similar to [Walsh, 2000]
- new atoms $e(v, i)$, v is assigned the value $i \in \{1, \dots, d\}$, $v = i$
 $\{e(v, 1), \dots, e(v, d)\} \leftarrow$
 $\perp \leftarrow \text{not } e(v, 1), \dots, \text{not } e(v, d)$
 $\perp \leftarrow 2 \{e(v, 1), \dots, e(v, d)\}$
- encode forbidden combination of values $v_1 = i_1, v_2 = i_2, \dots, v_n = i_n$:
 $\text{violate}(c) \leftarrow e(v_1, i_1), e(v_2, i_2), \dots, e(v_n, i_n)$

Theorem

Unit-propagation on the direct encoding does not guarantee arc consistency on the binary decomposition of the original constraint.

A binary constraint c is **arc consistent** iff a variable v is assigned any value there exist compatible values for the other variable in $\text{scope}(c)$.

Support Encoding

- similar to [Gent, 2002]

A **support** for $v = i$ across constraint c is the set of possible assignments $w = j_1, \dots, w = j_m$ to another variable w in the scope of the constraint that allow $v = i$.

- encode variables as in the direct encoding
- whenever $v = i$, at least one of its supports must hold:

$$\text{violate}(c) \leftarrow e(v, i), \text{not } e(w, j_1), \dots, \text{not } e(w, j_m)$$

Support Encoding

- similar to [Gent, 2002]

A **support** for $v = i$ across constraint c is the set of possible assignments $w = j_1, \dots, w = j_m$ to another variable w in the scope of the constraint that allow $v = i$.

- encode variables as in the direct encoding
- whenever $v = i$, at least one of its supports must hold:

$$\text{violate}(c) \leftarrow e(v, i), \text{not } e(w, j_1), \dots, \text{not } e(w, j_m)$$

Theorem

Unit-propagation on the support encoding enforces arc consistency on the binary decomposition of the original constraint.

Bound Encoding

- new propositional variables $b(v, i)$, representing $v \leq i$
- similar to order encoding [Tamura et. al, 2006]
- consistent set of bounds, $v \leq i \Rightarrow v \leq i + 1$

$$\{b(v, 1), \dots, b(v, d)\} \leftarrow$$

$$\perp \leftarrow b(v, i), \text{ not } b(v, i + 1)$$

$$\perp \leftarrow \text{ not } b(v, d)$$

- encode conflict regions $l_1 < v_1 \leq u_1, \dots, l_n < v_n \leq u_n$:

$$\text{violate}(c) \leftarrow b(v_1, u_1), \dots, b(v_n, u_n), \text{ not } b(v_1, l_1), \dots, \text{ not } b(v_n, l_n)$$

Bound Encoding

- new propositional variables $b(v, i)$, representing $v \leq i$
- similar to order encoding [Tamura et. al, 2006]
- consistent set of bounds, $v \leq i \Rightarrow v \leq i + 1$

$$\{b(v, 1), \dots, b(v, d)\} \leftarrow$$

$$\perp \leftarrow b(v, i), \text{ not } b(v, i + 1)$$

$$\perp \leftarrow \text{ not } b(v, d)$$

- encode conflict regions $l_1 < v_1 \leq u_1, \dots, l_n < v_n \leq u_n$:

$$\text{violate}(c) \leftarrow b(v_1, u_1), \dots, b(v_n, u_n), \text{ not } b(v_1, l_1), \dots, \text{ not } b(v_n, l_n)$$

Theorem

Unit-propagation on the bound encoding enforces bound consistency on the original constraint.

Constraint c is **bound consistent** iff a variable v is assigned $\min(\text{dom}(v))$ or $\max(\text{dom}(v))$ there exist compatible values between the minimum and maximum value for all other variables in $\text{scope}(c)$, a **bound support**.

Range Encoding

- new propositional variables $r(v, l, u)$, representing $v \in [l, u]$
- consistent ranges, $v \in [l, u] \Rightarrow v \in [l - 1, u] \wedge v \in [l, u + 1]$

$$r(v, l, u) \leftarrow \text{not } r(v, l - 1, u), \text{not } r(v, u + 1, u)$$

$$\perp \leftarrow r(v, l, u - 1), \text{not } r(v, l, u)$$

$$\perp \leftarrow r(v, l + 1, u), \text{not } r(v, l, u)$$

- encode conflict regions $v_1 \in [l_1, u_1], \dots, v_n \in [l_n, u_n]$:

$$\text{violate}(c) \leftarrow r(v_1, l_1, u_1), \dots, r(v_n, l_n, u_n)$$

Range Encoding

- new propositional variables $r(v, l, u)$, representing $v \in [l, u]$
- consistent ranges, $v \in [l, u] \Rightarrow v \in [l - 1, u] \wedge v \in [l, u + 1]$

$$r(v, l, u) \leftarrow \text{not } r(v, l - 1, u), \text{not } r(v, u + 1, u)$$

$$\perp \leftarrow r(v, l, u - 1), \text{not } r(v, l, u)$$

$$\perp \leftarrow r(v, l + 1, u), \text{not } r(v, l, u)$$

- encode conflict regions $v_1 \in [l_1, u_1], \dots, v_n \in [l_n, u_n]$:

$$\text{violate}(c) \leftarrow r(v_1, l_1, u_1), \dots, r(v_n, l_n, u_n)$$

Theorem

Unit-propagation on the range encoding enforces range consistency on the original constraint.

A constraint is **range consistent** iff when a variable is assigned any value in its domain, there exists a bound support.

Corollary

Unit-propagation on our support encoding of global ALL-DIFFERENT enforces arc consistency on the original constraint in $\mathcal{O}(nd^2)$ down any branch of the search tree.

Corollary

Unit-propagation on our bound encoding of global ALL-DIFFERENT enforces bound consistency on the original constraint in $\mathcal{O}(nd^2)$ down any branch of the search tree.

Corollary

Unit-propagation on our range encoding of global ALL-DIFFERENT enforces range consistency on the original constraint in $\mathcal{O}(nd^3)$ down any branch of the search tree.

- Benchmark domain: constraint satisfaction problems (CSP) containing ALL-DIFFERENT and PERMUTATION constraints
- Systems: CLINGCON; EZCSP; GECODE; INCA using
 - support encoding (S),
 - bound encoding (B, B_k), and
 - range encoding (R, R_k)for global constraints only (otherwise, direct encoding)
- Setting: 2.00 GHz PC under Linux, each run limited to 600 s time and 1 GB RAM

Quasigroup Completion

20 × 20

%	S	B_3	B	R_3	R	EZCSP	CLINGCON	GECODE	GECODE _{BC}
10	2.6	5.0	8.2	6.0	7.3	29.6 (7)	9.7 (4)	2.2 (4)	0.5 (1)
20	2.4	5.0	8.0	6.2	7.2	21.3 (20)	6.2 (5)	5.0 (4)	0.9 (3)
30	2.3	4.8	7.9	6.1	7.1	10.3 (30)	12.9 (13)	2.9 (13)	1.1 (5)
35	2.3	4.8	7.9	6.1	7.0	21.6 (24)	11.2 (17)	14.1 (13)	6.2 (7)
40	2.3	4.7	7.8	6.0	6.9	51.6 (29)	23.1 (22)	11.7 (20)	5.7 (9)
45	2.3	4.7	7.8	5.9	6.8	36.3 (35)	14.7 (28)	17.7 (25)	6.3 (13)
50	2.3	4.6	7.7	5.9	6.8	36.1 (50)	21.2 (37)	25.1 (32)	6.3 (18)
55	2.3	4.5	7.6	5.8	6.7	61.4 (51)	24.4 (44)	19.6 (41)	30.9 (29)
60	2.2	4.4	7.5	5.6	6.6	60.2 (63)	31.4 (56)	36.0 (51)	27.2 (35)
70	2.2	4.2	7.1	5.1	6.0	70.0 (66)	30.2 (50)	28.0 (45)	17.0 (27)
80	2.1	4.0	6.7	4.7	5.5	16.2 (18)	4.2 (18)	17.2 (13)	7.0 (7)
90	2.1	4.0	6.7	4.7	5.5	1.4	2.6 (1)	0.4 (1)	3.2

- phase transition behaviour of the hybrid CASP and pure CP systems
- conflict-driven learning exploits constraint interdependencies which improve propagation between constraints

Further Experiments and Conclusions

- see paper for more benchmarks

Contribution:

- translation-based approach to incorporating CP techniques into ASP
- generic ASP encodings for constraints s.t. unit-propagation enforces a certain level of consistency
- techniques are formulated as preprocessing and can be applied to any ASP solver without changing its source code
- promising results from experiments

- attractive, fun, and interesting area

Challenge

Enhancing Answer Set Programming (ASP) by Constraint Programming (CP) techniques, towards

- improved modelling convenience, and
- improved solving speed.

$$\text{CASP} = \text{ASP} + \text{CP}$$

- encoding of further constraints, e.g., REACHABILITY
- tackling space complexity though generating ASP encoding lazily; similar to [Ohrimenko et. al, 2009]