Systems and Solving Techniques for Knowledge Representation

– Cautious Reasoning –

Marco Maratea
University of Genoa, Italy

066 011 Double degree programme Computational Logic (Erasmus-Mundus)
066 931 Computational Intelligence
066 937 Software Engineering & Internet Computing
Institute of Information Systems

ASP: Cautious Reasoning

A *program* Π consists of finitely many rules of the form

$$a \leftarrow b_1, \ldots, b_l, not\ b_{l+1}, \ldots not\ b_m$$

where

- the *head* a is an atom or $\bot$, and
- in the *body* each $b_i (1 \leq i \leq m)$ is an atom.

Answer sets defined in terms of *reduct* and minimality [Gelfond and Lifschitz, 1988].

Cautious reasoning: solutions must be witnessed by all answer sets.

ASP solvers for cautious reasoning

ASP solvers DLV and CLASP have been extended with devoted techniques to deal with cautious reasoning tasks on top of their procedures for computing answer sets.

Main Idea: Starting from an over- and an under-approximation of the solution, solutions are searched via calls to ASP oracles to improve over-approximation.

[Alviano et al., 2014] presented a unified view of such solving procedures, and designed several algorithms for cautious reasoning in ASP.

[Alviano et al., 2014] also included other techniques borrowed from backbone computation of CNF formulas, and implemented all these techniques in WASP [Alviano et al., 2013].

ASP solvers for cautious reasoning

ASP solvers DLV and CLASP have been extended with devoted techniques to deal with cautious reasoning tasks on top of their procedures for computing answer sets.

Main Idea: Starting from an over- and an under-approximation of the solution, solutions are searched via calls to ASP oracles to improve over-approximation.

[Alviano et al., 2014] presented a unified view of such solving procedures, and designed several algorithms for cautious reasoning in ASP.

[Alviano et al., 2014] also included other techniques borrowed from backbone computation of CNF formulas, and implemented all these techniques in WASP [Alviano et al., 2013].

ASP solvers for cautious reasoning

ASP solvers DLV and CLASP have been extended with devoted techniques to deal with cautious reasoning tasks on top of their procedures for computing answer sets.

Main Idea: Starting from an over- and an under-approximation of the solution, solutions are searched via calls to ASP oracles to improve over-approximation.

[Alviano et al., 2014] presented a unified view of such solving procedures, and designed several algorithms for cautious reasoning in ASP.

[Alviano et al., 2014] also included other techniques borrowed from backbone computation of CNF formulas, and implemented all these techniques in WASP [Alviano et al., 2013].

ASP solvers for cautious reasoning

ASP solvers DLV and CLASP have been extended with devoted techniques to deal with cautious reasoning tasks on top of their procedures for computing answer sets.

Main Idea: Starting from an over- and an under-approximation of the solution, solutions are searched via calls to ASP oracles to improve over-approximation.

[Alviano et al., 2014] presented a unified view of such solving procedures, and designed several algorithms for cautious reasoning in ASP.

[Alviano et al., 2014] also included other techniques borrowed from backbone computation of CNF formulas, and implemented all these techniques in WASP [Alviano et al., 2013].

Abstract solvers for computing answer sets

We assume that the graph (V_X, *oracle*), where:

- V_X is the set of states related to a set X of atoms, and
- *oracle* is a set of transition rules,

describes the behavior of a general backtracking-based ASP solvers.

To find an answer set of a program Π it is enough to find a path in ($V_{atoms(\Pi)}$, *oracle*) leading from a proper initial node ($\emptyset$) to a terminal node ($Ok(L)$, $atoms(L) \subseteq X$), employing the *oracle* set of transition rules.

Abstract solvers for computing answer sets

We assume that the graph $(V_X, oracle)$, where:

- V_X is the set of states related to a set X of atoms, and
- *oracle* is a set of transition rules,

describes the behavior of a general backtracking-based ASP solvers.

To find an answer set of a program Π it is enough to find a path in $(V_{atoms(\Pi)}, oracle)$ leading from a proper initial node ($\emptyset$) to a terminal node ($Ok(L)$, $atoms(L) \subseteq X$), employing the *oracle* set of transition rules.

Intuition about the states

Intermediate states

The core states $L_{O,U,A}$ and the control states $Cont(O, U)$ represent all the intermediate steps of the computation; they are such that:

- L is the current state of the computation of a model;
- O is the current over-approximation of the solution stored as a set;
- U is the current under-approximation of the solution stored as a set;
- A is the action currently carried out: *init* if we search for a first model, *over* (resp. $under_{\{l\}}$) action if over-approximation (resp. under-approximation on a literal l) is being applied.

Intuition

- A core state $L_{O,U,A}$ represents the computation within a call to an ASP oracle, while
- a control state $Cont(O, U)$ controls the computation among different calls to ASP oracles.

Initial state

The initial state is $\emptyset_{atoms(\Pi),\emptyset,init}$.

States

Actions

For a set of atoms X, an *action relative to X* is an element of the set $\{init, over\} \cup \{under_{\{l\}} | l \in lit(X)\}$.

States

The set of *states relative to X*, written V_X, is the union of:

- The set of *core states relative to X*, that are all $L_{O,U,A}$, s.t. L is a record relative to X, O and U are sets of literals relative to X, and A is an action relative to X.
- The set of *control states relative to X*, that are all the $Cont(O, U)$ where O and U are sets of literals relative to X.
- The *fail state UNSAT*;
- The set of *final states relative to X*, that are all the $Ok(W)$ where W is a set of literals relative to X.

Modeling over-approximation

Return rules				
$Fail_{over}$	$L_{O,U,over}$	$\Longrightarrow$	$Ok(O)$	if $\{$ L is inconsistent and decision-free
$Find$	$L_{O,U,A}$	$\Longrightarrow$	$Cont(O \cap L, U)$	if $\{$ no other return/oracle rule applies
Control rules				
$Success$	$Cont(O, O)$	$\Longrightarrow$	$Ok(O)$	
$OverApprox$	$Cont(O, U)$	$\Longrightarrow$	$\emptyset_{O,U,over}$	if $\{$ $O \neq U$

Figure : The transition rules of *ov*.

Modeling over-approximation (II)

We define $\Pi_{O,U,over}$ as $\Pi \cup \{\leftarrow O\}$.

For any Π, the graph OS_{Π} is $(V_{atoms(\Pi)}, oracle \cup ov)$ abstracts Algorithm A2 of [Alviano et al., 2014].

Formal result

For any program Π, if a terminal state $Ok(W)$ is reached in OS_{Π} from the initial state, then W is the intersection of all answer sets of Π. Otherwise, *UNSAT* is reached and Π does not have answer sets.

Modeling under-approximation

Return rule

$$
\begin{array}{llll}
\mathit{Fail}_{under} & L_{O,U,under_{\{l\}}} & \Longrightarrow \mathit{Cont}(O \setminus \{\bar{l}\}, U \cup \{l\}) & \text{if } \left\{ \begin{array}{l} L \text{ is inconsistent and} \\ \text{decision-free} \end{array} \right. \\
\mathit{Find} & L_{O,U,A} & \Longrightarrow \mathit{Cont}(O \cap L, U) & \text{if } \left\{ \begin{array}{l} \text{no other return/oracle} \\ \text{rule applies} \end{array} \right.
\end{array}
$$

Control rule

$$
\begin{array}{llll}
\mathit{Success} & \mathit{Cont}(O, O) & \Longrightarrow \mathit{Ok}(O) & \\
\mathit{UnderApprox} & \mathit{Cont}(O, U) & \Longrightarrow \emptyset_{O,U,under_{\{l\}}} & \text{if } \{ \; l \in O \setminus U
\end{array}
$$

Figure : The transition rules of *un*.

Modeling under-approximation (II)

We define $\Pi_{O,U,under_l}$ as $\Pi \cup \{\leftarrow l\}$.

For any Π, the graph US_Π is $(V_{atoms(\Pi)}, oracle \cup un)$. abstracts Algorithm A3 of [Alviano et al., 2014].

Formal result

For any program Π, if a terminal state $Ok(W)$ is reached in US_Π from the initial state, then W is the intersection of all answer sets of Π. Otherwise, $UNSAT$ is reached and Π does not have answer sets.

Mixing over-and under-approximation

For any Π, the graph $MixS_{\Pi}$ is $(V_{atoms(\Pi)}, oracle \cup un \cup ov)$. abstracts Algorithm A1 of [Alviano et al., 2014].

Formal result

For any program Π, if a terminal state $Ok(W)$ is reached in $MixS_{\Pi}$ from the initial state, then W is the intersection of all answer sets of of Π. Otherwise, *UNSAT* is reached and Π does not have answer sets.

Full example

$$\Pi = \Pi_{\{a,b,c\},\emptyset,init} = \{$$
$$\leftarrow a, b$$
$$a \leftarrow \neg a, \neg b$$
$$a \leftarrow b$$
$$b \leftarrow \neg a, \neg b$$
$$b \leftarrow b$$
$$c \leftarrow \}$$

$$\Pi_{\{a,c\},\emptyset,over} = \Pi \cup \{$$
$$\leftarrow a, c \}$$

$$\Pi_{\{c\},\emptyset,under_{\{c\}}} = \Pi \cup \{$$
$$\leftarrow c$$
$$\leftarrow \neg c \}$$

$\emptyset_{\{a,b,c\},\emptyset,init}$

$UnitPropagate$:	$c_{\{a,b,c\},\emptyset,init}$
$Decide$:	$ca^{\Delta}{}_{\{a,b,c\},\emptyset,init}$
$UnitPropagate$:	$ca^{\Delta}\neg b_{\{a,b,c\},\emptyset,init}$
$Find$:	$Cont(\{a,c\},\emptyset)$
$OverApprox$:	$\emptyset_{\{a,c\},\emptyset,over}$
$UnitPropagate$:	$c_{\{a,c\},\emptyset,over}$
$UnitPropagate$:	$c\neg a_{\{a,c\},\emptyset,over}$
$UnitPropagate$:	$c\neg ab_{\{a,c\},\emptyset,over}$
$Find$:	$Cont(\{c\},\emptyset)$
$UnderApprox$:	$\emptyset_{\{c\},\emptyset,under_{\{c\}}}$
$UnitPropagate$:	$c_{\{c\},\emptyset,under_{\{c\}}}$
$UnitPropagate$:	$c\neg c_{\{c\},\emptyset,under_{\{c\}}}$
$Fail_{under}$:	$Cont(\{c\},\{c\})$
$Success$:	$Ok(\{c\})$

References I

Alviano, M., Dodaro, C., Faber, W., Leone, N., and Ricca, F. (2013).
WASP: A native ASP solver based on constraint learning.
In Cabalar, P. and Son, T. C., editors, *Proceedings of the 12th International Conference of Logic Programming and Nonmonotonic Reasoning (LPNMR 2013)*, volume 8148 of *Lecture Notes in Computer Science*, pages 54–66. Springer.

Alviano, M., Dodaro, C., and Ricca, F. (2014).
Anytime computation of cautious consequences in answer set programming.
Theory and Practive of Logic Programming, 14(4-5):755–770.

Gelfond, M. and Lifschitz, V. (1988).
The stable model semantics for logic programming.
In Kowalski, R. and Bowen, K., editors, *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP 1988)*, pages 1070–1080. MIT Press.

APPENDIX