Systems and Solving Techniques for Knowledge Representation

– (Disjunctive) ASP solving –

Marco Maratea
University of Genoa, Italy

066 011 Double degree programme Computational Logic
066 931 Computational Intelligence
066 937 Software Engineering & Internet Computing
Institute of Information Systems

Goal of this part

In this lecture the goal is to show how some solvers for disjunctive ASP solving implementing plain backtracking, i.e.

- CMODELS,
- GNT,
- DLV,

try to solve the program at hand.

Again, we employ abstract solvers for presenting algorithm's behavior.

A Two Layers Solver Architecture

A common architecture of a disjunctive answer set solver is composed of two layers: a *generate* layer and a *test* layer.

- The generate layer is used to obtain a set of candidates that are potentially answer sets of a given program.
- The test layer is used to verify whether a candidate is indeed an answer set of the program.

A Two Layers Solver Architecture: Idea

Idea

By taking advantage of the two layers architecture, the idea is to design abstract solvers made of two graphs (with the modeling seen for non-disjunctive ASP), that “communicate” each other via novel transition rules that model the outcomes of their respective solving processes.

A Two Layers Abstract Solver Architecture: States

A *state* relative to sets X and X' of atoms is either

1. a pair $(L, R)_s$, where L and R are records relative to X and X', respectively, and s is a label ($s \in \{\mathcal{L}, \mathcal{R}\}$).
2. $Ok(L)$, where L is a record relative to X, or
3. the distinguished state *UNSAT*.

Disjunctive programs

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

$$a_1 \vee \dots \vee a_n \leftarrow b_1, \dots, b_l, not\ b_{l+1}, \dots not\ b_m$$

where

- in the *head*

  $$a_1 \vee \dots \vee a_n$$

  each a_j is an atom, and n can be 0 ($\perp$), and
- in the *body*

  $$b_1, \dots, b_l, not\ b_{l+1}, \dots not\ b_m$$

  each $b_i (1 \leq i \leq m)$ is an atom.

We can identify a rule with the clause

$$a_1 \vee \ldots \vee a_n \vee \overline{b_1} \vee \ldots \vee \overline{b_l} \vee b_{l+1} \vee \ldots \vee b_m.$$

A Two Layers Abstract Solver Architecture: Notation

Covering

We say that a set M of literals *covers* a program Π if $atoms(\Pi) \subseteq atoms(M)$.

Generating function

A function g from a program to another program is a *generating (program)* function if for any program Π, $atoms(\Pi) \subseteq atoms(g(\Pi))$.

Witness function

A function $t(\Pi, L)$ from Π and a consistent set L of literals covering Π to a non-disjunctive program Π' is called *witness (program)* function.

For a witness function t, $atoms(t, \Pi, X)$ denotes the union of $atoms(t(\Pi, L))$ for all possible consistent and complete sets L of literals over X.

Abstract disjunctive CMODELS: Graph $DPLL^2_{g,t}(\Pi)$

In CMODELS, the generate and test layers are SAT oracles.

$DPLL^2_{g,t}(\Pi)$ graph

- Its nodes consists of the states relative to sets $atoms(g(\Pi))$ and $atoms(t, \Pi, atoms(g(\Pi)))$, and
- its edges are described by modified (wrt $DPLL_F$) and additional transition rules.

Initial state

The initial state is $(\emptyset, \emptyset)_{\mathcal{L}}$.

Graph $DPLL^2_{g,t}(\Pi)$: Transition rules (I)

Left-rules

$$
\begin{array}{llll}
Conclude_{\mathcal{L}} & (L,\emptyset)_{\mathcal{L}} & \Longrightarrow UNSAT & \text{if} \begin{cases} L \text{ is inconsistent and} \\ L \text{ contains no decision literal} \end{cases} \\
Backtrack_{\mathcal{L}} & (Ll^{\Delta}L',\emptyset)_{\mathcal{L}} & \Longrightarrow (L\bar{l},\emptyset)_{\mathcal{L}} & \text{if} \begin{cases} Ll^{\Delta}L' \text{ is inconsistent and} \\ L' \text{ contains no decision literal} \end{cases} \\
Unit_{\mathcal{L}} & (L,\emptyset)_{\mathcal{L}} & \Longrightarrow (Ll,\emptyset)_{\mathcal{L}} & \text{if} \begin{cases} l \text{ is a literal over } atoms(g(\Pi)) \text{ and} \\ l \text{ does not occur in } L \text{ and} \\ \text{a rule in } g(\Pi) \text{ is equivalent to } C \vee l \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } L \end{cases} \\
Decide_{\mathcal{L}} & (L,\emptyset)_{\mathcal{L}} & \Longrightarrow (Ll^{\Delta},\emptyset)_{\mathcal{L}} & \text{if} \begin{cases} L \text{ is consistent and} \\ l \text{ is a literal over } atoms(g(\Pi)) \text{ and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } L \end{cases}
\end{array}
$$

Figure : The transition rules of the graph $DPLL^2_{g,t}(\Pi)$.

Graph $DPLL^2_{g,t}(\Pi)$: Transition rules (II)

Right-rules

$$\begin{array}{llll}
Conclude_{\mathcal{R}} & (L,R)_{\mathcal{R}} & \Longrightarrow Ok(L) & \text{if } \begin{cases} R \text{ is inconsistent and} \\ R \text{ contains no decision literal} \end{cases} \\[2ex]
Backtrack_{\mathcal{R}} & (L,Rl^{\Delta}R')_{\mathcal{R}} & \Longrightarrow (L,R\bar{l})_{\mathcal{R}} & \text{if } \begin{cases} Rl^{\Delta}R' \text{ is inconsistent and} \\ R' \text{ contains no decision literal} \end{cases} \\[2ex]
Unit_{\mathcal{R}} & (L,R)_{\mathcal{R}} & \Longrightarrow (L,Rl)_{\mathcal{R}} & \text{if } \begin{cases} l \text{ is a literal over } atoms(t(\Pi,L)) \text{ and} \\ l \text{ does not occur in } R \text{ and} \\ \text{a rule in } t(\Pi,L) \text{ is equivalent to } C \vee l \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } L \end{cases} \\[2ex]
Decide_{\mathcal{R}} & (L,R)_{\mathcal{R}} & \Longrightarrow (L,Rl^{\Delta})_{\mathcal{R}} & \text{if } \begin{cases} R \text{ is consistent and} \\ l \text{ is a literal over } atoms(t(\Pi,L)) \text{ and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } R \end{cases}
\end{array}$$

Figure : The transition rules of the graph $DPLL^2_{g,t}(\Pi)$.

Graph $DPLL^2_{g,t}(\Pi)$: Transition rules (III)

Crossing-rule $\mathcal{LR}$

$$Cross_{\mathcal{LR}} \quad (L, \emptyset)_{\mathcal{L}} \quad \Longrightarrow (L, \emptyset)_{\mathcal{R}} \quad \text{if } \left\{ \text{no left-rule applies} \right.$$

Crossing-rules $\mathcal{RL}$

$$Conclude_{\mathcal{RL}} \quad (L, R)_{\mathcal{R}} \quad \Longrightarrow UNSAT \quad \text{if } \begin{cases} \text{no right-rule applies and} \\ L \text{ contains no decision literal} \end{cases}$$

$$Backtrack_{\mathcal{RL}} \quad (Ll^{\Delta}L', R)_{\mathcal{R}} \Longrightarrow (L\bar{l}, \emptyset)_{\mathcal{L}} \quad \text{if } \begin{cases} \text{no right-rule applies and} \\ L' \text{ contains no decision literal} \end{cases}$$

Figure : The transition rules of the graph $DPLL^2_{g,t}(\Pi)$.

A definition for the next formal results

Definition

We say that a graph G *checks* the stable models of a program Π when all the following conditions hold:

1. G is finite and acyclic;
2. Any terminal state in G is either *UNSAT* or of the form $Ok(L)$;
3. If a state $Ok(L)$ is reachable from the initial state in G then $L_{|atoms(\Pi)}$ is an answer s of Π;
4. *UNSAT* is reachable from the initial state in G if and only if Π does not have answer sets.

Abstract disjunctive CMODELS

CMODELS with plain backtracking implements DP-ASSAT-PROC procedure [Lierler, 2005].

Given a disjunctive program Π, in CMODELS:

- the generate layer relies on $g^C(\Pi)$, which corresponds to the clausified $Comp(\Pi)$, and
- the test layer relies on a witness formula function t^C that intuitively tests minimality of models of completion.

These functions are defined, e.g. at pages 11-12 of [Brochenin et al., 2016].

Abstract disjunctive CMODELS: Formal result

Theorem

For any program Π, *the graph* $DPLL^2_{g^C,t^C}(\Pi)$ *checks the stable models of* Π.

Abstract GNT: Graph

In GNT [Janhunen et al., 2006], instead, the two layers employ instances of SMODELS.

$SM^2_{g,t}(\Pi)$ graph

- The nodes are defined as previously, and
- the edges are justified by the transition rules of SM_Π, marked with subscript $s \in \{\mathcal{L}, \mathcal{R}\}$, and crossing rules of $DPLL^2_{g,t}(\Pi)$ graph.

Abstract GnT: Formal result

We are given

- $g^G(\Pi)$, and
- $t^G(\Pi, L)$

defined as in [Janhunen et al., 2006].

Theorem

For any Π, the graph $SM^2_{g^G,t^G}(\Pi)$ checks the stable models of Π.

Abstract GnT: Example (I)

Let

$$Propagate$$

correspond to the application of a transition rule in $\{UnitPropagateLP, AllRulesCancelled, BackchainTrue, BackchainFalse, Unfounded\}$ and

$$Propagate^n$$

refer to the application of (the same rule in) *Propagate* (n times, $n > 1$).

Given the following program Π:

$$\begin{array}{c} a \leftarrow c. \\ b \leftarrow c. \\ c \leftarrow a, b. \\ a \vee b \leftarrow . \end{array}$$

Abstract GNT: Example (II)

$$
g^G(\Pi) = \begin{array}{l}
a \leftarrow c \\
b \leftarrow c \\
c \leftarrow a, b \\
a \leftarrow not\ a^r \\
b \leftarrow not\ b^r \\
a^r \leftarrow not\ a \\
b^r \leftarrow not\ b \\
\leftarrow not\ a, not\ b \\
a^s \leftarrow c \\
a^s \leftarrow not\ b \\
b^s \leftarrow c \\
b^s \leftarrow not\ a \\
\leftarrow a, not\ a^s \\
\leftarrow b, not\ b^s
\end{array}
$$

$$
t^G(\Pi, L) = \begin{array}{l}
a \leftarrow not\ a^r \\
a^r \leftarrow not\ a \\
\leftarrow not\ a, not\ b \\
\leftarrow a, not\ b, not\ c
\end{array}
$$

$$
\begin{array}{ll}
\text{Initial state}: & (\emptyset, \emptyset)_{\mathcal{L}} \\
Decide_{\mathcal{L}}: & ((\overline{a^r})^\Delta, \emptyset)_{\mathcal{L}} \\
Propagate^2_{\mathcal{L}}: & ((\overline{a^r})^\Delta\ a\ a^s, \emptyset)_{\mathcal{L}} \\
Decide_{\mathcal{L}}: & ((\overline{a^r})^\Delta\ a\ a^s\ \overline{b}^\Delta, \emptyset)_{\mathcal{L}} \\
Propagate_{\mathcal{L}}: & ((\overline{a^r})^\Delta\ a\ a^s\ \overline{b}^\Delta\ b^r, \emptyset)_{\mathcal{L}} \\
Propagate_{\mathcal{L}}: & ((\overline{a^r})^\Delta\ a\ a^s\ \overline{b}^\Delta\ b^r\ \overline{c}, \emptyset)_{\mathcal{L}} \\
Decide_{\mathcal{L}}: & ((\overline{a^r})^\Delta\ a\ a^s\ \overline{b}^\Delta\ b^r\ \overline{c}\ b^s, \emptyset)_{\mathcal{L}} \\
Cross_{\mathcal{L}\mathcal{R}}: & ((\overline{a^r})^\Delta\ a\ a^s\ \overline{b}^\Delta\ b^r\ \overline{c}, \emptyset)_{\mathcal{R}}
\end{array}
$$

Let $L = (\overline{a^r})^\Delta\ a\ a^s\ \overline{b}^\Delta\ b^r\ \overline{c}$

$$
\begin{array}{ll}
\text{Current state}: & (L, \emptyset)_{\mathcal{R}} \\
Decide_{\mathcal{R}}: & (L, \overline{a}^\Delta)_{\mathcal{R}} \\
Propagate_{\mathcal{R}}: & (L, \overline{a}^\Delta\ b)_{\mathcal{R}} \\
Propagate_{\mathcal{R}}: & (L, \overline{a}^\Delta\ b\ \overline{b})_{\mathcal{R}} \\
Backtrack_{\mathcal{R}}: & (L, a)_{\mathcal{R}} \\
Propagate_{\mathcal{R}}: & (L, a\ \overline{a^r})_{\mathcal{R}} \\
Propagate^2_{\mathcal{R}}: & (L, a\ \overline{a^r}\ \overline{b}\ \overline{c})_{\mathcal{R}} \\
Propagate_{\mathcal{R}}: & (L, a\ \overline{a^r}\ \overline{b}\ \overline{c}\ c)_{\mathcal{R}} \\
Conclude_{\mathcal{R}}: & Ok(L)
\end{array}
$$

Abstract DLV: Graph

In DLV [Leone et al., 2006], the generate layer is similar to an application of SMODELS, while the test layer employs a SAT solver.

$SM_g^\vee(\Pi) \times DPLL_t(\Pi)$ graph

- The nodes are defined as previously, and
- the edges are justified by the transition rules of $DPLL_F$, marked with $\mathcal{R}$, crossing rules, and modified SM_Π (called $SM_\Pi^\vee$) rules, marked with $\mathcal{L}$, without *Unfounded* and with some updated left rules, e.g.

$$\textit{dAllRulesCancelled}_{\mathcal{L}}:$$

$$(L, \emptyset)_{\mathcal{L}} \Longrightarrow (L\bar{a}, \emptyset)_{\mathcal{L}} \text{ if } \begin{cases} \text{for each rule } a \vee A \leftarrow B \text{ of } \Pi \\ B \text{ is contradicted by } L \end{cases}$$

Abstract DLV: Formal result

We are given

- $g^D(\Pi)$ to be the identity function, and
- $t^D(\Pi, L)$ defined as in [Koch et al., 2003].

Theorem

For any Π, the graph $SM^{\vee}_{g^D}(\Pi) \times DPLL_{t^D}(\Pi)$ checks the stable models of Π.

Abstract DLV: Example

Given the following program Π:

$$
\begin{array}{c}
a \leftarrow c. \\
b \leftarrow c. \\
c \leftarrow a, b. \\
a \vee b \leftarrow .
\end{array}
$$

Abstract DLV: Example (II)

$$g^D(\Pi) = \begin{array}{l} a \leftarrow c \\ b \leftarrow c \\ c \leftarrow a, b \\ a \vee b \leftarrow \end{array}$$

$$t^D(\Pi, L) = \begin{array}{l} \overline{a} \vee c \\ \overline{b} \vee c \\ \overline{c} \vee a \vee b \\ \overline{a} \vee \overline{b} \\ c \vee a \vee b \end{array}$$

$$t^D(\Pi, L') = \begin{array}{l} \overline{a} \\ \overline{b} \\ \overline{c} \vee a \\ \overline{a} \vee \overline{b} \\ a \end{array}$$

$$\begin{array}{ll} \text{Initial state}: & (\emptyset, \emptyset)_{\mathcal{L}} \\ Decide_{\mathcal{L}}: & (c^{\Delta}, \emptyset)_{\mathcal{L}} \\ Propagate^2_{\mathcal{L}}: & (c^{\Delta}\ a\ b, \emptyset)_{\mathcal{L}} \\ Cross_{\mathcal{L}\mathcal{R}}: & (c^{\Delta}\ a\ b, \emptyset)_{\mathcal{R}} \end{array}$$

Let $L = c^{\Delta}\ a\ b$

$$\begin{array}{ll} \text{Current state}: & (L, \emptyset)_{\mathcal{R}} \\ Decide_{\mathcal{R}}: & (L, a^{\Delta})_{\mathcal{R}} \\ Propagate^2_{\mathcal{R}}: & (L, a^{\Delta}\ c\ \overline{b})_{\mathcal{R}} \\ Backtrack_{\mathcal{R}\mathcal{L}}: & (\overline{c}, \emptyset)_{\mathcal{L}} \\ Decide_{\mathcal{L}}: & (\overline{c}\ a^{\Delta}, \emptyset)_{\mathcal{L}} \\ Propagate_{\mathcal{L}}: & (\overline{c}\ a^{\Delta}\ \overline{b}, \emptyset)_{\mathcal{L}} \\ Cross_{\mathcal{L}\mathcal{R}}: & (\overline{c}\ a^{\Delta}\ \overline{b}, \emptyset)_{\mathcal{R}} \end{array}$$

Let $L' = \overline{c}\ a^{\Delta}\ \overline{b}$

$$\begin{array}{ll} \text{Current state}: & (L', \emptyset)_{\mathcal{R}} \\ Propagate^2_{\mathcal{R}}: & (L', \overline{a}\ a)_{\mathcal{R}} \\ Conclude_{\mathcal{R}}: & Ok(L') \end{array}$$

Designing a new abstract solver through combination

New $DPLL_g(\Pi) \times SM_t(\Pi)$ graph

- The set of nodes are defined as previously.
- The edges of the graph $DPLL_g(\Pi) \times SM_t(\Pi)$ are specified by (*i*) the Left-rules and Crossing-rules of the graph $DPLL^2_{g,t}$, and (*ii*) the Right-rules $SM^2_{g,t}$.

References I

Brochenin, R., Lierler, Y., and Maratea, M. (2014).
Abstract disjunctive answer set solvers.
In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 165–170. IOS Press.

Brochenin, R., Lierler, Y., and Maratea, M. (To appear in 2016).
Abstract disjunctive answer set solvers via templates.
Theory and Practice of Logic Programming. Available at `http://www.star.dist.unige.it/~marco/Data/16tplp.pdf`.

Janhunen, T., Niemelä, I., Seipel, D., Simons, P., and You, J.-H. (2006).
Unfolding partiality and disjunctions in stable model semantics.
ACM Transactions on Computunational Logic, 7(1):1–37.

Koch, C., Leone, N., and Pfeifer, G. (2003).
Enhancing disjunctive logic programming systems by sat checkers.
Artificial Intelligence, 151:177–212.

References II

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2006).
The DLV system for knowledge representation and reasoning.
ACM Transactions on Computational Logic, 7(3):499–562.

Lierler, Y. (2005).
Cmodels: SAT-based disjunctive answer set solver.
In Baral, C., Greco, G., Leone, N., and Terracina, G., editors, *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005)*, volume 3662 of *Lecture Notes in Computer Science*, pages 447–452.