Abstract Solvers for Answer Set Programming

Marco Maratea[1]

thanks: Yuliya Lierler[2], Remi Brochenin[1]

[1] DIBRIS, University of Genoa, Italy
[2] Dep. of Computer Science, University of Nebraska at Omaha

ICLP 2015: Cork, Ireland, Sept 2nd 2015

Motivation

Issue

- Usually solving algorithms are presented by means of pseudo-code descriptions, but
- some communities have experienced that analyzing such algorithms on this basis may not be fruitful.

Instead ...

- more formal descriptions, based on mathematically precise but possibly simple objects, can be useful, and
- can allow for, e.g. a uniform representation.

Motivation

Issue

- Usually solving algorithms are presented by means of pseudo-code descriptions, but
- some communities have experienced that analyzing such algorithms on this basis may not be fruitful.

Instead ...

- more formal descriptions, based on mathematically precise but possibly simple objects, can be useful, and
- can allow for, e.g. a uniform representation.

Abstract solvers

Abstract solvers are a relatively new methodology for describing, comparing and composing solving procedures in an abstract way via graphs, where

- the states of computation are represented as nodes,
- the solving techniques as edges between such nodes,
- the solving process as a path in the graph, and
- formal properties of the procedures are reduced to related graph's properties.

What are they good for?

- Describing abstract solving procedures in a clear mathematical and unified way via graphs.
- Comparing solving techniques employed in different procedures by means of comparison of related graphs.
- Combining abstract solving procedures, by means of modular addition/deletion of techniques/edges, to design novel abstract procedures.

What are they not (that) good for?

- Specifying (low level) implementation details.
- Arguing about the efficiency of an implementation built on this basis.

Outline

Outline

DPLL SAT solving: Some notation

Given a set X of atoms,

- a record relative to X is a string L composed of literals over X or the symbol $\bot$, with no repetitions.
- Some literal l, called *decision literal*, may be annotated as l^{Δ}.

(In)Consistent records

We say that a record L is *inconsistent* if it contains both a literal l and its complement $\bar{l}$, or if it contains $\bot$, and *consistent* otherwise.

DPLL SAT solving: Some notation

Given a set X of atoms,

- a record relative to X is a string L composed of literals over X or the symbol $\bot$, with no repetitions.
- Some literal l, called *decision literal*, may be annotated as l^{Δ}.

(In)Consistent records

We say that a record L is *inconsistent* if it contains both a literal l and its complement $\bar{l}$, or if it contains $\bot$, and *consistent* otherwise.

DPLL algorithm for SAT solving

The Davis-Putnam-Logemann-Loveland algorithm [Davis and Putnam, 1960, Davis et al., 1962] is the most famous and used (backtracking-based) algorithm for solving the propositional satisfiability (SAT) problem.

Propositional formulas are in Conjunctive Normal Form (CNF), i.e. set of clauses.

A *model* of a CNF formula F is a (total) assignment to variables satisfying F.

DPLL SAT solving: Nodes of the graph $DPLL_F$

A *state* relative to (a set of atoms) X is either

1. a record relative to X, or
2. the distinguished state *SAT* or *UNSAT*.

Nodes of the graph $DPLL_F$

- The set of nodes of graph $DPLL_F$ consists of the states relative to the set of atoms $atoms(F)$ appearing in F.
- A node in the graph is terminal if no edge originates from it.
- The state $\emptyset$ is called initial.

DPLL SAT solving: Nodes of the graph $DPLL_F$

A *state* relative to (a set of atoms) X is either

1. a record relative to X, or
2. the distinguished state *SAT* or *UNSAT*.

Nodes of the graph $DPLL_F$

- The set of nodes of graph $DPLL_F$ consists of the states relative to the set of atoms $atoms(F)$ appearing in F.
- A node in the graph is terminal if no edge originates from it.
- The state $\emptyset$ is called initial.

DPLL SAT solving: Edges of the graph $DPLL_F$

Conclude :	$L \Longrightarrow \mathit{UNSAT}$	if L is inconsistent and L contains no decision literals
Backtrack :	$Ll^{\Delta}L' \Longrightarrow L\bar{l}$	if $Ll^{\Delta}L'$ is inconsistent and L' contains no decision literals
Unit :	$L \Longrightarrow Ll$	if l does not occur in L and F contains a clause $C \vee l$ and all the literals of $\overline{C}$ occur in L
Decide :	$L \Longrightarrow Ll^{\Delta}$	if L is consistent and neither l nor $\bar{l}$ occur in L
Success :	$L \Longrightarrow \mathit{SAT}$	if no other rule applies

Figure : Transition rules that justify edges of the graph $DPLL_F$.

DPLL SAT solving: Formal result

[Nieuwenhuis et al., 2006]; Proposition 1 in [Lierler, 2011]

Theorem

For any CNF formula F,

1. *graph $DPLL_F$ is finite and acyclic,*
2. *any terminal state reachable from $\emptyset$ in $DPLL_F$ other than $UNSAT$ is SAT, and*
3. *$UNSAT$ is reachable from $\emptyset$ in $DPLL_F$ if and only if F is unsatisfiable.*

DPLL SAT solving: Examples

Initial state :		$\emptyset$
Decide	$\Longrightarrow$	a^{Δ}
Unit	$\Longrightarrow$	$a^{\Delta}\ c$
Decide	$\Longrightarrow$	$a^{\Delta}\ c\ b^{\Delta}$
Success	$\Longrightarrow$	*SAT*

Initial state :		$\emptyset$
Decide	$\Longrightarrow$	a^{Δ}
Decide	$\Longrightarrow$	$a^{\Delta}\ \overline{c}^{\Delta}$
Unit	$\Longrightarrow$	$a^{\Delta}\ \overline{c}^{\Delta}\ c$
Backtrack	$\Longrightarrow$	$a^{\Delta}\ c$
Decide	$\Longrightarrow$	$a^{\Delta}\ c\ b^{\Delta}$
Success	$\Longrightarrow$	*SAT*

Figure : Examples of paths in $DPLL_{\{a \vee b,\ \overline{a} \vee c\}}$.

CDCL algorithm for SAT

The Conflict-Driven Clause Learning algorithm for SAT "extends" the DPLL algorithm with optimized backtracking techniques, i.e. *backjumping* and *learning* borrowed from CSP [Prosser, 1993].

See, e.g. [Bayardo and Schrag, 1997, Marques-Silva and Sakallah, 1996, Zhang et al., 2001]

CDCL SAT solving: Extended states

$DPLLearn_F$ graph

1. Its nodes are extended states relative to F, and
2. its edges are justified by extended, updated and additional transition rules wrt $DPLL_F$.

For a CNF formula F, an *extended state* relative to F is either

1. a pair (L, Γ), written $L||\Gamma$, where
   - L is a record relative to $atoms(F)$, and
   - Γ is a set of clauses over $atoms(F)$ that are entailed by F;
   or
2. the distinguished state *SAT* or *UNSAT*.

Initial state

The (extended) initial state is $\emptyset||\emptyset$.

CDCL SAT solving: Extended states

$DPLLearn_F$ graph

1. Its nodes are extended states relative to F, and
2. its edges are justified by extended, updated and additional transition rules wrt $DPLL_F$.

For a CNF formula F, an *extended state* relative to F is either

1. a pair (L, Γ), written $L||\Gamma$, where
   - L is a record relative to $atoms(F)$, and
   - Γ is a set of clauses over $atoms(F)$ that are entailed by F;
   or
2. the distinguished state *SAT* or *UNSAT*.

Initial state

The (extended) initial state is $\emptyset||\emptyset$.

CDCL SAT solving: Updated and extended rules (I)

Conclude : $L \| \Gamma \Longrightarrow \mathit{UNSAT}$ if $\begin{cases} L \text{ is inconsistent and} \\ L \text{ contains no decision literals} \end{cases}$

Backjump : $L l^{\Delta} L' \| \Gamma \Longrightarrow L l' \| \Gamma$ if $\begin{cases} L l^{\Delta} L' \text{ is inconsistent and} \\ F \models l' \vee \overline{L} \end{cases}$

UnitLearn : $L \| \Gamma \Longrightarrow L l \| \Gamma$ if $\begin{cases} l \text{ does not occur in } L \text{ and} \\ F \cup \Gamma \text{ contains a clause } C \vee l \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } L \end{cases}$

Decide : $L \| \Gamma \Longrightarrow L l^{\Delta} \| \Gamma$ if $\begin{cases} L \text{ is consistent and} \\ \text{neither } l \text{ nor } \overline{l} \text{ occur in } L \end{cases}$

CDCL SAT solving: Additional transition rules

$$\textit{Learn}: \qquad L\|\Gamma \Longrightarrow L\|C \cup \Gamma \quad \text{if} \begin{cases} \text{every atom in } C \text{ occurs in } F \text{ and} \\ F \models C \end{cases}$$

CDCL SAT solving: Updated and extended rules (II)

Conclude : $L \| \Gamma \Longrightarrow \mathit{UNSAT}$ if $\begin{cases} L \text{ is inconsistent and} \\ L \text{ contains no decision literals} \end{cases}$

Backjump : $L l^{\Delta} L' \| \Gamma \Longrightarrow L l' \| \Gamma$ if $\begin{cases} L l^{\Delta} L' \text{ is inconsistent and} \\ F \models l' \vee \overline{L} \end{cases}$

UnitLearn : $L \| \Gamma \Longrightarrow L l \| \Gamma$ if $\begin{cases} l \text{ does not occur in } L \text{ and} \\ F \cup \Gamma \text{ contains a clause } C \vee l \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } L \end{cases}$

Decide : $L \| \Gamma \Longrightarrow L l^{\Delta} \| \Gamma$ if $\begin{cases} L \text{ is consistent and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } L \end{cases}$

Success : $L \| \Gamma \Longrightarrow \mathit{SAT}$ if no other rule applies other then *Learn*

CDCL SAT solving: Additional transition rules (II)

$$\textit{Learn}: \qquad L \| \Gamma \Longrightarrow L \| C \cup \Gamma \quad \text{if} \begin{cases} \text{every atom in } C \text{ occurs in } F \text{ and} \\ F \models C \end{cases}$$

CDCL SAT solving: Additional transition rules (II)

Learn : $L||\Gamma \Longrightarrow L||C \cup \Gamma$ if $\begin{cases} \text{every atom in } C \text{ occurs in } F \text{ and} \\ F \models C \end{cases}$

Restart : $L||\Gamma \Longrightarrow \emptyset||\Gamma$

Forget : $L||C \cup \Gamma \Longrightarrow L||\Gamma$

CDCL SAT solving: Example

$$
\begin{array}{lcl}
\text{Initial state :} & & \emptyset \| \emptyset \\
\textit{Learn} & \Longrightarrow & \emptyset \| \{b \vee c\} \\
\textit{Decide} & \Longrightarrow & \overline{b}^{\Delta} \| \{b \vee c\} \\
\textit{UnitLearn} & \Longrightarrow & \overline{b}^{\Delta} c \| \{b \vee c\} \\
\textit{UnitLearn} & \Longrightarrow & \overline{b}^{\Delta} c\ a \| \{b \vee c\} \\
\textit{Success} & \Longrightarrow & \textit{SAT}
\end{array}
$$

Figure : Example of path in $DPLLearn_{\{a \vee b,\ \overline{a} \vee c\}}$.

DPLL SAT solving: States (slightly modified)

A *state* relative to (a set of atoms) X is either

1. A record relative to X,
2. $Ok(L)$ where L is a record relative to X, or
3. The distinguished state *UNSAT*.

States and graphs

- The set of nodes of $DPLL_F$ consists of the states relative to the set of atoms appearing in F $atoms(F)$.
- A node in the graph is *terminal* if no edge originates from it.
- The state $\emptyset$ is called *initial*.
- Each formula F determines its DPLL graph $DPLL_F$.

DPLL SAT solving: States (slightly modified)

A *state* relative to (a set of atoms) X is either

1. A record relative to X,
2. $Ok(L)$ where L is a record relative to X, or
3. The distinguished state *UNSAT*.

States and graphs

- The set of nodes of $DPLL_F$ consists of the states relative to the set of atoms appearing in F $atoms(F)$.
- A node in the graph is *terminal* if no edge originates from it.
- The state $\emptyset$ is called *initial*.
- Each formula F determines its DPLL graph $DPLL_F$.

DPLL SAT solving: Transition rules (slightly modified)

Conclude : $L \Longrightarrow \mathit{UNSAT}$ if $\begin{cases} L \text{ is inconsistent and} \\ L \text{ contains no decision literals} \end{cases}$

Backtrack : $Ll^{\Delta}L' \Longrightarrow L\bar{l}$ if $\begin{cases} Ll^{\Delta}L' \text{ is inconsistent and} \\ L' \text{ contains no decision literals} \end{cases}$

Unit : $L \Longrightarrow Ll$ if $\begin{cases} l \text{ does not occur in } L \text{ and} \\ F \text{ contains a clause } C \vee l \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } L \end{cases}$

Decide : $L \Longrightarrow Ll^{\Delta}$ if $\begin{cases} L \text{ is consistent and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } L \end{cases}$

Success : $L \Longrightarrow \mathit{Ok}(L)$ if no other rule applies

DPLL SAT solving: Formal result (slightly modified)

Theorem

For any CNF formula F,

1. *graph $DPLL_F$ is finite and acyclic,*
2. *any terminal state reachable from $\emptyset||\emptyset$ in $DPLL_F$ other than UNSAT is $Ok(L)$, with (the assignment that can be built from) L being a model of F, and*
3. *UNSAT is reachable from $\emptyset$ in $DPLL_F$ if and only if F is unsatisfiable.*

DPLL SAT solving: Formal result (slightly modified)

Theorem

For any CNF formula F,

1. *graph $DPLL_F$ is finite and acyclic,*
2. *any terminal state reachable from $\emptyset$ in $DPLL_F$ other than UNSAT is $Ok(L)$, with L being a model of F, and*
3. *UNSAT is reachable from $\emptyset$ in $DPLL_F$ if and only if F is unsatisfiable.*

Outline

Non-disjunctive programs

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

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

where

- the *head* a is an atom or $\bot$, and
- in the *body* $b_1, \ldots, b_l, \mathit{not}\ b_{l+1}, \ldots \mathit{not}\ b_m$, each $b_i (1 \leq i \leq m)$ is an atom.

We can identify a rule with the clause

$$a \vee \overline{b_1} \vee \ldots \vee \overline{b_l} \vee b_{l+1} \vee \ldots \vee b_m$$

and also with the set of its elements.

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

SAT-based Generate&Test procedure [Lierler, 2008]

We first present a modification of the $DPLL_F$ graph.

Setting

- F is a CNF formula,
- G is formula formed from atoms in $atoms(F)$.

Graph $GT_{F,G}$

- The nodes are the same as $DPLL_F$.
- The edges are justified by the transition rules of $DPLL_F$ and

$$Test: \quad L \Longrightarrow L\bar{l} \quad \text{if} \begin{cases} L \text{ is consistent and} \\ G \models \overline{L} \text{ and} \\ l \in L \end{cases}$$

SAT-based Generate&Test procedure: Formal result

Theorem

For any CNF formula F and a formula G formed from atoms(F)

1. *graph $GT_{F,G}$ is finite and acyclic,*
2. *any terminal state reachable from $\emptyset$ in $GT_{F,G}$ other than UNSAT is Ok(L), with L being a model of $F \wedge G$, and*
3. *UNSAT is reachable from $\emptyset$ in $GT_{F,G}$ if and only if $F \wedge G$ is unsatisfiable.*

Comparing solving procedures through graphs

At the beginning of this tutorial, we have mentioned that solving procedures can be conveniently compared through the study of their related graphs.

As an example, it is easy to see that the graph $DPLL_F$ is a subgraph of $GT_{F,G}$.

Comparing solving procedures through graphs

At the beginning of this tutorial, we have mentioned that solving procedures can be conveniently compared through the study of their related graphs.

As an example, it is easy to see that the graph $DPLL_F$ is a subgraph of $GT_{F,G}$.

Abstract CMODELS with backtracking

Given a logic program Π, consider

- the (plain) CNF conversion of the completion $Comp(\Pi)$, and
- the conjunction of all loop formulas of Π, $LF(\Pi)$.

Abstract CMODELS with backtracking

- $GT_{Comp(\Pi),LF(\Pi)}$ abstracts CMODELS with backtracking implementing ASP-SAT procedure [Giunchiglia et al., 2006], by applying *Test* only on models of F.
- If the state $Ok(L)$ is reached, then the set of atoms in L, L^+, is an answer set of Π.

Abstract CMODELS with backtracking: Example

Given the following program Π,

$$a \leftarrow a.$$

Initial state :		$\emptyset$
Decide	$\Longrightarrow$	a^{Δ}
Test	$\Longrightarrow$	$a^{\Delta}\overline{a}$
Backtrack	$\Longrightarrow$	$\overline{a}$
Success	$\Longrightarrow$	$Ok(\overline{a})$

Figure : Example of path in $GT_{\{a \vee \overline{a}, \overline{a}\}}$.

$\{\overline{a}\}^{+} = \emptyset$ is an (the only) answer set of Π.

G&T with learning: Extended GT state and graph

For a CNF formula F, and a formula G formed from atoms $atoms(F)$, an *extended (GT) state* relative to F and G is either

1. a pair (L, Γ), written $L||\Gamma$, where
   - L is a record relative to $atoms(F)$, and
   - Γ is a set of clauses over $atoms(F)$ that are entailed by $F \wedge G$; or
2. the distinguished state $Ok(L)$ or $UNSAT$.

$GTL_{F,G}$ graph

1. Its nodes are extended GT states relative to F and G, and
2. its transition rules are *UnitLearn*, *Decide*, *Conclude*, *Success* of $DPLLearn_F$, plus the three following rules.

G&T with learning: Extended and Updated rules

BackjumpGT :
$$Ll^{\Delta}L' \| \Gamma \Longrightarrow Ll' \| \Gamma \quad \text{if} \begin{cases} Ll^{\Delta}L' \text{ is inconsistent and} \\ F \wedge G \models l' \vee \overline{L} \end{cases}$$

LearnGT :
$$L \| \Gamma \Longrightarrow L \| C \cup \Gamma \quad \text{if} \begin{cases} \text{every atom in } C \text{ occurs in } F \text{ and} \\ F \wedge G \models C \end{cases}$$

Test :
$$L \| \Gamma \Longrightarrow L\bar{l} \| \Gamma \quad \text{if} \begin{cases} L \text{ is consistent and} \\ G \models \overline{L} \text{ and} \\ l \in L \end{cases}$$

G&T with learning: Formal result

Theorem

For any CNF formula F and a formula G formed from atoms(F)

1. *every path in $GTL_{F,G}$ uses only finitely many times edges justified by transition rules other than Learn,*
2. *any terminal state reachable from $\emptyset$ in $GTL_{F,G}$ other than UNSAT is Ok(L), with L being a model of $F \wedge G$, and*
3. *UNSAT is reachable from $\emptyset$ in $GT_{F,G}$ if and only if $F \wedge G$ is unsatisfiable.*

Abstract CMODELS

Given a logic program Π, if

- F is the CNF conversion of the completion $Comp(\Pi)$, and
- G is $LF(\Pi)$,

Abstract CMODELS with learning

- $GTL_{Comp(\Pi),LF(\Pi)}$ abstracts CMODELS with learning [Lierler, 2005] implementing ASP-SAT procedure+learning [Giunchiglia et al., 2006], by
  1. applying *LearnGT* in a state reached by the application of *BackjumpGT*, and
  2. assigning priorities to the application of the transition rules as follows: *BackjumpGT*, *Conclude* $>>$ *UnitLearn* $>>$ *Decide* $>>$ *Test*. Such ordering guarantees that
     - *Test* is applied only on models of $F \cup \Gamma$, and
     - *BackjumpGT* is first applied on a state reached by the application of *Test*.
- If the state $Ok(L)$ is reached, then L^+ is an answer set of Π.

Abstract CMODELS

Given a logic program Π, if

- F is the CNF conversion of the completion $Comp(\Pi)$, and
- G is $LF(\Pi)$,

Abstract CMODELS with learning

- $GTL_{Comp(\Pi),LF(\Pi)}$ abstracts CMODELS with learning [Lierler, 2005] implementing ASP-SAT procedure+learning [Giunchiglia et al., 2006], by
  1. applying *LearnGT* in a state reached by the application of *BackjumpGT*, and
  2. assigning priorities to the application of the transition rules as follows: *BackjumpGT*, *Conclude* $>>$ *UnitLearn* $>>$ *Decide* $>>$ *Test*. Such ordering guarantees that
     - *Test* is applied only on models of $F \cup \Gamma$, and
     - *BackjumpGT* is first applied on a state reached by the application of *Test*.
- If the state $Ok(L)$ is reached, then L^+ is an answer set of Π.

Abstract CLASP

CLASP [Gebser et al., 2007]

- Employs an additional rule wrt $GTL_{F,G}$:

$$\textit{Unfounded}: \quad L \Longrightarrow L\overline{a} \quad \text{if} \begin{cases} L \text{ is consistent and} \\ a \in U \text{ for a set } U \text{ unfounded on } L \text{ w.r.t. } \Pi \end{cases}$$

- Follows the ordering on rules application: *BackjumpGT*, *Conclude*$>>$ *UnitLearn*, *Unfounded* $>>$ *Decide*.
- Applies *LearnGT* in a state reached by the application of *BackjumpGT*.

Abstract $ATLEAST_{\Pi}$

We now define a graph whose terminal nodes correspond to supported models of a program Π.

$ATLEAST_{\Pi}$ graph

1. Its nodes are the states relative to the set of atoms $atoms(\Pi)$, and
2. its edges are justified by the transition rules *Decide*, *Conclude*, *Backtrack*, *Success* of the *DPLL* graph, and some additional rules that describe deterministic consequences.

Abstract $ATLEAST_{\Pi}$

We now define a graph whose terminal nodes correspond to supported models of a program Π.

$ATLEAST_{\Pi}$ graph

1. Its nodes are the states relative to the set of atoms $atoms(\Pi)$, and
2. its edges are justified by the transition rules *Decide*, *Conclude*, *Backtrack*, *Success* of the *DPLL* graph, and some additional rules that describe deterministic consequences.

Abstract $ATLEAST_{\Pi}$: Additional rules

UnitPropagateLP :
$$L \Longrightarrow La \quad \text{if} \quad \begin{cases} \text{there is a rule } a \leftarrow B \text{ of } \Pi \text{ such that} \\ \quad B \subseteq L \end{cases}$$

AllRulesCancelled :
$$L \Longrightarrow L\overline{a} \quad \text{if} \quad \begin{cases} \text{for each rule } a \leftarrow B \text{ of } \Pi \\ \quad B \text{ is contradicted by } L \end{cases}$$

BackchainTrue :
$$L \Longrightarrow Ll \quad \text{if} \quad \begin{cases} \text{there is a rule } a \leftarrow l, B \text{ of } \Pi \text{ such that} \\ \quad a \text{ is in } L \text{ and} \\ \text{for each other rule } a \leftarrow B' \text{ of } \Pi \\ \quad B' \text{ is contradicted by } L \end{cases}$$

BackchainFalse :
$$L \Longrightarrow L\overline{l} \quad \text{if} \quad \begin{cases} \text{there is a rule } a \leftarrow l, B \text{ of } \Pi \text{ such that} \\ \quad \overline{a} \text{ is in } L \text{ or } a = \bot \text{ and} \\ \quad B \subseteq L \end{cases}$$

Abstract $ATLEAST_\Pi$: Formal result

Theorem

For any program Π*,*

1. *graph* $ATLEAST_\Pi$ *is finite and acyclic,*
2. *any terminal state reachable from* $\emptyset$ *in* $ATLEAST_\Pi$ *other than UNSAT is* $Ok(L)$*, with* L *being a supported model of* Π*, and*
3. *UNSAT is reachable from* $\emptyset$ *in* $ATLEAST_\Pi$ *if and only if* Π *has no supported models.*

Abstract $ATLEAST_{\Pi}$: Formal result (II)

Theorem [Lierler, 2011]

For any program Π, the graphs $ATLEAST_{\Pi}$ and $DPLL_{Comp(\Pi)}$ are equal.

Abstract $ATLEAST_{\Pi}$: Example

Let Π be the following program:

$$a \leftarrow not\ b.$$
$$b \leftarrow not\ a.$$
$$c \leftarrow a.$$
$$d \leftarrow d.$$

Initial state :		$\emptyset$
Decide	$\Longrightarrow$	a^{Δ}
UnitPropagateLP	$\Longrightarrow$	$a^{\Delta}c$
AllRulesCancelled	$\Longrightarrow$	$a^{\Delta}c\overline{b}$
Decide	$\Longrightarrow$	$a^{\Delta}c\overline{b}d^{\Delta}$
Success	$\Longrightarrow$	$Ok(a^{\Delta}c\overline{b}d^{\Delta})$

Figure : Example of path in $ATLEAST_{\Pi}$

Abstract SMODELS: SM_Π Graph

SM_Π graph

- Its nodes are the same as of the graph $ATLEAST_\Pi$, and
- its edges are justified by the transition rules of $ATLEAST_\Pi$ and *Unfounded*

Unfounded : $L \Longrightarrow L\bar{a}$ if $\begin{cases} L \text{ is consistent and} \\ a \in U \text{ for a set } U \text{ unfounded on } L \text{ w.r.t. } \Pi \end{cases}$

SM_Π: Formal result

Theorem

For any program Π*,*

1. *graph* SM_Π *is finite and acyclic,*
2. *any terminal state reachable from* $\emptyset$ *in* SM_Π *other than UNSAT is* $Ok(L)$*, with* L^+ *being an answer set of* Π*, and*
3. *UNSAT is reachable from* $\emptyset$ *in* SM_Π *if and only if* Π *has no answer sets.*

SM_{Π}: Example

Let Π be the following program:

$$a \leftarrow not\ b.$$
$$b \leftarrow not\ a.$$
$$c \leftarrow a.$$
$$d \leftarrow\ d.$$

Initial state :		$\emptyset$
Decide	$\Longrightarrow$	a^{Δ}
UnitPropagateLP	$\Longrightarrow$	$a^{\Delta}c$
AllRulesCancelled	$\Longrightarrow$	$a^{\Delta}c\overline{b}$
Decide	$\Longrightarrow$	$a^{\Delta}c\overline{b}d^{\Delta}$
...	...	...

$\{a, c, \overline{b}, d\}$ is a supported model of Π, but not an answer set.

SM_Π: Example (II)

Let Π be the following program:

$$a \leftarrow not\ b.$$
$$b \leftarrow not\ a.$$
$$c \leftarrow a.$$
$$d \leftarrow\ d.$$

Initial state :		$\emptyset$
Decide	$\Longrightarrow$	a^Δ
UnitPropagateLP	$\Longrightarrow$	$a^\Delta c$
AllRulesCancelled	$\Longrightarrow$	$a^\Delta c\overline{b}$
Decide	$\Longrightarrow$	$a^\Delta c\overline{b}d^\Delta$
Unfounded	$\Longrightarrow$	$a^\Delta c\overline{b}d^\Delta\overline{d}$
Backtrack	$\Longrightarrow$	$a^\Delta c\overline{b}\ \overline{d}$
Success	$\Longrightarrow$	$Ok(a^\Delta c\overline{b}\ \overline{d})$

Figure : Example of path in SM_Π.

Abstract SMODELS via SM_Π

SMODELS [Simons et al., 2002] priorities

Backtrack, *Conclude* $>>$
UnitPropagateLP, *AllRulesCancelled*, *BackchainTrue*, *BackchainFalse* $>>$
Unfounded $>>$ *Decide*.

Initial state :		$\emptyset$
Decide	$\Longrightarrow$	a^{Δ}
UnitPropagateLP	$\Longrightarrow$	$a^{\Delta} c$
AllRulesCancelled	$\Longrightarrow$	$a^{\Delta} c\overline{b}$
Unfounded	$\Longrightarrow$	$a^{\Delta} c\overline{b}\ \overline{d}$
Success	$\Longrightarrow$	$Ok(a^{\Delta} c\overline{b}\ \overline{d})$

Figure : Example of path followed by SMODELS on Π.

Abstract SMODELS$_{cc}$: Graph SML_Π (Idea)

[Ward and Schlipf, 2004]

For a program Π, an *extended state* relative to Π is either

1. a pair (L, Γ), written $L||\Gamma$, where
   - L is a record relative to $atoms(\Pi)$, and
   - Γ is a set of constraints over $atoms(\Pi)$ that are entailed by Π; or
2. the distinguished state $Ok(L)$ or $UNSAT$.

SML_Π graph

- Its nodes are the extended states relative to Π, and
- its edges are justified by extended, updated and additional transition rules wrt SM_Π.

SUP: A new solver with few changes [Lierler, 2008]

The graph of SUP_Π is a subgraph of SM_Π with

- the same nodes, and
- the same transition rules but *Unfounded*, which is now

$$\textit{UnfoundedSUP}: \quad L\|\Gamma \Longrightarrow L\bar{a}\|\Gamma \quad \text{if} \begin{cases} \text{no atom is unassigned by } L \\ L \text{ is consistent and} \\ a \in U \text{ for a set } U \text{ unfounded on } L \end{cases}$$

In [Lierler, 2011] SUP_Π has been extended with backjumping and learning rules of SML_Π, with the following priorities:
BackjumpLP, *Conclude* >>
UnitPropagateLP, *AllRulesCancelled*, *BackchainTrue*, *BackchainFalse* >>
Decide >> *Unfounded*.

The implementation of SUP led to positive results. SUP participated to the 3rd ASP Competition.

SUP: A new solver with few changes [Lierler, 2008]

The graph of SUP_{Π} is a subgraph of SM_{Π} with

- the same nodes, and
- the same transition rules but *Unfounded*, which is now

$$\textit{UnfoundedSUP}: \quad L||\Gamma \Longrightarrow L\overline{a}||\Gamma \quad \text{if} \begin{cases} \text{no atom is unassigned by } L \\ L \text{ is consistent and} \\ a \in U \text{ for a set } U \text{ unfounded on } L \end{cases}$$

In [Lierler, 2011] SUP_{Π} has been extended with backjumping and learning rules of SML_{Π}, with the following priorities:
BackjumpLP, *Conclude* >>
UnitPropagateLP, *AllRulesCancelled*, *BackchainTrue*, *BackchainFalse* >>
Decide >> *Unfounded*.

The implementation of SUP led to positive results. SUP participated to the 3rd ASP Competition.

SUP: A new solver with few changes [Lierler, 2008]

The graph of SUP_{Π} is a subgraph of SM_{Π} with

- the same nodes, and
- the same transition rules but *Unfounded*, which is now

$$\textit{UnfoundedSUP}: \quad L\|\Gamma \Longrightarrow L\bar{a}\|\Gamma \quad \text{if} \begin{cases} \text{no atom is unassigned by } L \\ L \text{ is consistent and} \\ a \in U \text{ for a set } U \text{ unfounded on } L \end{cases}$$

In [Lierler, 2011] SUP_{Π} has been extended with backjumping and learning rules of SML_{Π}, with the following priorities:
BackjumpLP, *Conclude* $>>$
UnitPropagateLP, *AllRulesCancelled*, *BackchainTrue*, *BackchainFalse* $>>$
Decide $>>$ *Unfounded*.

The implementation of SUP led to positive results. SUP participated to the 3rd ASP Competition.

Outline

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 before), 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

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

where

- in the *head*

  $$a_1 \vee \cdots \vee a_n$$

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

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

  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

$$\textit{Conclude}_{\mathcal{L}} \quad (L, \emptyset)_{\mathcal{L}} \Longrightarrow \textit{UNSAT} \quad \text{if} \begin{cases} L \text{ is inconsistent and} \\ L \text{ contains no decision literal} \end{cases}$$

$$\textit{Backtrack}_{\mathcal{L}} \quad (Ll^{\Delta}L', \emptyset)_{\mathcal{L}} \Longrightarrow (L\bar{l}, \emptyset)_{\mathcal{L}} \quad \text{if} \begin{cases} Ll^{\Delta}L' \text{ is inconsistent and} \\ L' \text{ contains no decision literal} \end{cases}$$

$$\textit{Unit}_{\mathcal{L}} \quad (L, \emptyset)_{\mathcal{L}} \Longrightarrow (Ll, \emptyset)_{\mathcal{L}} \quad \text{if} \begin{cases} l \text{ is a literal over } \textit{atoms}(g(\Pi)) \text{ and} \\ l \text{ does not occur in } L \text{ and} \\ \text{a rule in } g(\Pi) \text{ is equivalent to } C \vee l \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } L \end{cases}$$

$$\textit{Decide}_{\mathcal{L}} \quad (L, \emptyset)_{\mathcal{L}} \Longrightarrow (Ll^{\Delta}, \emptyset)_{\mathcal{L}} \quad \text{if} \begin{cases} L \text{ is consistent and} \\ l \text{ is a literal over } \textit{atoms}(g(\Pi)) \text{ and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } L \end{cases}$$

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

Right-rules

$$\textit{Conclude}_{\mathcal{R}} \quad (L, R)_{\mathcal{R}} \quad \Longrightarrow \textit{Ok}(L) \quad \text{if} \begin{cases} R \text{ is inconsistent and} \\ R \text{ contains no decision literal} \end{cases}$$

$$\textit{Backtrack}_{\mathcal{R}} \quad (L, Rl^{\Delta}R')_{\mathcal{R}} \Longrightarrow (L, R\bar{l})_{\mathcal{R}} \quad \text{if} \begin{cases} Rl^{\Delta}R' \text{ is inconsistent and} \\ R' \text{ contains no decision literal} \end{cases}$$

$$\textit{Unit}_{\mathcal{R}} \quad (L, R)_{\mathcal{R}} \quad \Longrightarrow (L, Rl)_{\mathcal{R}} \quad \text{if} \begin{cases} l \text{ is a literal over } \textit{atoms}(t(\Pi, L)) \text{ and} \\ l \text{ does not occur in } R \text{ and} \\ \text{a rule in } t(\Pi, L) \text{ is equivalent to } C \vee l \text{ and} \\ \text{all the literals of } \overline{C} \text{ occur in } L \end{cases}$$

$$\textit{Decide}_{\mathcal{R}} \quad (L, R)_{\mathcal{R}} \quad \Longrightarrow (L, Rl^{\Delta})_{\mathcal{R}} \quad \text{if} \begin{cases} R \text{ is consistent and} \\ l \text{ is a literal over } \textit{atoms}(t(\Pi, L)) \text{ and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } R \end{cases}$$

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

Crossing-rule $\mathcal{LR}$

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

Crossing-rules $\mathcal{RL}$

$$\textit{Conclude}_{\mathcal{RL}} \quad (L, R)_{\mathcal{R}} \Longrightarrow \textit{UNSAT} \quad \text{if} \left\{ \begin{array}{l} \text{no right-rule applies and} \\ L \text{ contains no decision literal} \end{array} \right.$$

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

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 in [Lierler, 2011].

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 a *Propagate* rule (n times, $n > 1$).
Given the following program Π:

$$a \leftarrow c.$$
$$b \leftarrow c.$$
$$c \leftarrow a.$$
$$a \,|\, b \leftarrow .$$

Abstract GnT: Example (II)

$g^G(\Pi) =$

$a \leftarrow c$
$b \leftarrow c$
$c \leftarrow a, b$
$a \leftarrow not\ a^r$
$b \leftarrow not\ b^r$
$a^r \leftarrow not\ a$
$b^r \leftarrow not\ b$
$\leftarrow not\ a, not\ b$
$a^s \leftarrow c$
$a^s \leftarrow not\ b$
$b^s \leftarrow c$
$b^s \leftarrow not\ a$
$\leftarrow a, not\ a^s$
$\leftarrow b, not\ b^s$

$t^G(\Pi, L) =$

$a \leftarrow not\ a^r$
$a^r \leftarrow not\ a$
$\leftarrow not\ a, not\ b$
$\leftarrow a, not\ b, not\ c$

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

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

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

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.

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

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:

$$a \leftarrow c.$$
$$b \leftarrow c.$$
$$c \leftarrow a.$$
$$a \vee b \leftarrow .$$

Abstract DLV: Example (II)

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

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

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

Initial state :	$(\emptyset, \emptyset)_{\mathcal{L}}$
$Decide_{\mathcal{L}}$:	$(c^{\Delta}, \emptyset)_{\mathcal{L}}$
$Propagate^2_{\mathcal{L}}$:	$(c^{\Delta}\ a\ b, \emptyset)_{\mathcal{L}}$
$Cross_{\mathcal{LR}}$:	$(c^{\Delta}\ a\ b, \emptyset)_{\mathcal{R}}$

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

Current state :	$(L, \emptyset)_{\mathcal{R}}$
$Decide_{\mathcal{R}}$:	$(L, c^{\Delta})_{\mathcal{R}}$
$Propagate^2_{\mathcal{R}}$:	$(L, c^{\Delta}\ a\ b)_{\mathcal{R}}$
$Backtrack_{\mathcal{RL}}$:	$(\overline{c}, \emptyset)_{\mathcal{L}}$
$Decide_{\mathcal{L}}$:	$(\overline{c}\ a^{\Delta}, \emptyset)_{\mathcal{L}}$
$Propagate_{\mathcal{L}}$:	$(\overline{c}\ a^{\Delta}\ \overline{b}, \emptyset)_{\mathcal{L}}$
$Cross_{\mathcal{LR}}$:	$(\overline{c}\ a^{\Delta}\ \overline{b}, \emptyset)_{\mathcal{R}}$

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

Current state :	$(L', \emptyset)_{\mathcal{R}}$
$Propagate^2_{\mathcal{R}}$:	$(L', \overline{a}\ a)_{\mathcal{R}}$
$Conclude_{\mathcal{R}}$:	$Ok(L')$

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 $SM^2_{g,t}$.

Some current work on disjunctive ASP solvers [Brochenin et al., 2016] . . .

Given the common two layers architecture, we are

- generalizing the graphs to a graph template containing all transition rules,
- defining general conditions for generating and witness functions, such that
- the various abstract solvers are obtained by
  - properly instantiating such functions, and
  - selecting the proper transition rules.

. . . and if you want to stay on the very abstract side . . .

- An Abstract View on Modularity in Knowledge Representation [Lierler and Truszczynski, 2015]
- Abstract Modular Inference Systems and Solvers [Lierler and Truszczynski, 2014]
- Modular Answer Set Solving [Lierler and Truszczynski, 2013]

Outline

Abstract Answer Set Solvers for Cautious Reasoning

SOTA

- Some ASP solvers, e.g. CLASP and DLV, have been extended to compute cautious consequences of a program.
- [Alviano et al., 2014] presented several algorithms employing these and others techniques, implemented and tested in WASP [Alviano et al., 2013].

Abstract solvers for this task:

- Abstract procedures for all algorithms in [Alviano et al., 2014], and
- further algorithms by including techniques from backbone computation [Janota et al., 2015].

Abstract Answer Set Solvers for Cautious Reasoning

TALK on Friday at 10:30am

Abstract Answer Set Solvers for Cautious Reasoning

TALK on Friday at 10:30am

Outline

Other applications of Abstract Solvers

Abstract solvers have been employed in some other problems:

1. Quantified Boolean Formulas
2. Constraint Answer Set Programming
3. Abstract Argumentation Frameworks
4. Satisfiability Modulo Theories

Abstract Solvers for Quantified Boolean Formulas

- In [Brochenin and Maratea, 2015], three abstract solvers for solving QBFs are presented.
- One proposal abstracts the Q-DPLL algorithm for QBFs [Cadoli et al., 1998, Giunchiglia et al., 2009].
- Q-DPLL is an extension of the DPLL algorithm for SAT.

Q-DPLL for QBFs: Graph

We are given a (prenex CNF) QBF formula F.

QBF_F graph

- The nodes are the states defined similarly as for DPLL, but decision literals are either *universal* ($l^{\forall}$) or *existential* ($l^{\exists}$).
- The edges corresponds to updated and additional transition rules wrt DPLL graph.

Q-DPLL for QBFs: Transition rules

Unit	L	$\Longrightarrow$	Ll	if l does not occur in L and for some clause C in the formula, l occurs in C and each other unassigned literal of C is universal and each assigned literal of C is contradicted
*Monotone*1	L	$\Longrightarrow$	Ll	if the variable of l is existential and l occurs in some clause C and $\bar{l}$ does not occur in any clause C
*Monotone*2	L	$\Longrightarrow$	Ll	if the variable of l is universal and $\bar{l}$ occurs in some clause C and l does not occur in any clause C
Decide	L	$\Longrightarrow$	Ll^Q	if L is consistent and the variable of l is unassigned and the quantifier of the variable of l is Q and for all l' such that $level(l') < level(l)$ the variable of l' is assigned.
Backtrack$_\exists$	$Ll^\exists L'$	$\Longrightarrow$	$L\bar{l}$	if $Ll^\exists L'$ is inconsistent and $l^\exists$ is the rightmost existential literal
Fail	L	$\Longrightarrow$	*Unsat*	if L is inconsistent and existential free
Backtrack$_\forall$	$Ll^\forall L'$	$\Longrightarrow$	$L\bar{l}$	if no other rule applies except *Succeed* and $l^\forall$ is the rightmost universal literal
Succeed	L	$\Longrightarrow$	*Valid*	if no other rule applies

Figure : The transition rules of the QBF_F graph.

Abstract Solvers for Constraint ASP [Lierler, 2014]

Constraint answer set programming (CASP) integrates ASP with constraint processing.

In [Lierler, 2014], abstract solvers are employed to describe the following CASP solvers:

- ACSOLVER [Mellarkod et al., 2008]
- CLINGON [Ostrowski and Schaub, 2012]
- EZCSP [Balduccini, 2009]

Abstract Solvers for Abstract Argumentation Frameworks [Brochenin et al., 2015]

Abstract Argumentation Frameworks (AFs) is a dedicated AI formalism, defined by *a set of arguments* and *an attack relation*.

There exists a number of semantics for AFs, a number of reasoning tasks, and several systems [Charwat et al., 2015].

In [Brochenin et al., 2015], we have focused on the *preferred* semantic, and have designed abstract solvers for describing three algorithms for solving respective reasoning tasks

- ArgSemSat for preferred enumeration [Cerutti et al., 2014]
- CEGARTIX for skeptical acceptance [Dvořák et al., 2014]
- another approach for preferred enumeration [Nofal et al., 2014]

A new abstract solver is also proposed based on the "combination" of techniques in the first and third approach above.

Abstract Solvers for Satisfiability Modulo Theories

The original paper on abstract solvers [Nieuwenhuis et al., 2006] was not limited to SAT and CDCL SAT solving, but its ultimate goal was to present abstract solvers for SMT.

In [Nieuwenhuis et al., 2006] it is then defined an DPLL(T) abstract solver for an SMT T, based on a lazy/SAT-based architecture.

It was implemented in the BARCELOGIC SMT solver, and instantiated for some SMT theories, including Integer and Real Difference Logic.

BARCELOGIC won the SMT 2005 Competition on these theories.

Conclusions

In this tutorial I have overviewed abstract solvers in Answer Set Programming for

- both non-disjunctive and disjunctive programs, and
- different reasoning tasks.

Moreover, we have briefly touched the usage of abstract solvers in other problems.

Current and future directions

- Design abstract solvers for disjunctive ASP solvers implementing backjumping and learning.
- Design abstract solvers for QBF solvers implementing backjumping and learning.
- Extend/Compose current solvers with ideas borrowed from abstract solvers (e.g., ASP solvers for cautious reasoning; AF solvers).

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.

References II

Balduccini, M. (2009).

Representing constraint satisfaction problems in answer set programming.

in: Proceedings of ICLP Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP), https://www.mat.unical.it/ASPOCP09/.

Bayardo, R. and Schrag, R. (1997).

Using CSP look-back techniques to solve real-world SAT instances.

In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 203–208.

References III

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.

Accepted to Theory and Practice of Logic Programming.

Brochenin, R., Linsbichler, T., Maratea, M., Wallner, J. P., and Woltran, S. (2015).

Abstract solvers for dung's argumentation frameworks.

In *Accepted to the 2015 International Workshop on Theory and Applications of Formal Argument (TAFA 2015)*.

References IV

Brochenin, R. and Maratea, M. (2015).

Abstract solvers for quantified boolean formulas and their application.

In Springer, editor, *Proc. of the 14th Conference of the Italian Association for Artificial Intelligence*, volume LNAI.

To appear.

Cadoli, M., Giovanardi, A., and Schaerf, M. (1998).

An algorithm to evaluate quantified boolean formulae.

In Mostow, J. and Rich, C., editors, *Proc. of AAAI 1998*, pages 262–267. AAAI Press / The MIT Press.

References V

Cerutti, F., Dunne, P. E., Giacomin, M., and Vallati, M. (2014).

Computing preferred extensions in abstract argumentation: A SAT-based approach.

In Black, E., Modgil, S., and Oren, N., editors, *Proceedings of the Second International Workshop on Theory and Applications of Formal Argumentation, TAFA 2013*, volume 8306 of *Lecture Notes in Computer Science*, pages 176–193. Springer.

Charwat, G., Dvořák, W., Gaggl, S. A., Wallner, J. P., and Woltran, S. (2015).

Methods for Solving Reasoning Problems in Abstract Argumentation - A Survey.

Artificial Intelligence, 220:28–63.

References VI


Davis, M., Logemann, G., and Loveland, D. (1962).
A machine program for theorem proving.
Communications of the ACM, 5(7):394–397.


Davis, M. and Putnam, H. (1960).
A computing procedure for quantification theory.
Journal of ACM, 7:201–215.


Dvořák, W., Järvisalo, M., Wallner, J. P., and Woltran, S. (2014).
Complexity-sensitive decision procedures for abstract argumentation.
Artificial Intelligence, 206:53–78.


Gebser, M., Kaufmann, B., Neumann, A., and Schaub, T. (2007).
CLASP: A conflict-driven answer set solver.
In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*.

References VII

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.

Giunchiglia, E., Lierler, Y., and Maratea, M. (2006).

Answer set programming based on propositional satisfiability.

Journal of Automated Reasoning, 36:345–377.

Giunchiglia, E., Marin, P., and Narizzano, M. (2009).

Reasoning with quantified boolean formulas.

In Biere, A., Heule, M., van Maaren, H., and Walsh, T., editors, *Handbook of Satisfiability*, volume 185 of *FAIA*, pages 761–780. IOS Press.

References VIII


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.


Janota, M., Lynce, I., and Marques-Silva, J. (2015).
Algorithms for computing backbones of propositional formulae.
AI Communications, 28(2):161–177.


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


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.

References IX

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.

Lierler, Y. (2008).

Abstract answer set solvers.

In de la Banda, M. G. and Pontelli, E., editors, *Proceedings of the 24th International Conference on Logic Programming (ICLP 2008)*, volume 5366 of *Lecture Notes in Computer Science*, pages 377–391. Springer.

References X


Lierler, Y. (2011).
Abstract answer set solvers with backjumping and learning.
Theory and Practice of Logic Programming, 11:135–169.


Lierler, Y. (2014).
Relating constraint answer set programming languages and algorithms.
Artificial Intelligence, 207:1–22.


Lierler, Y. and Truszczynski, M. (2013).
Modular answer set solving.
In *Late-Breaking Developments in the Field of Artificial Intelligence*, volume WS-13-17 of *AAAI Workshops*. AAAI.

References XI

Lierler, Y. and Truszczynski, M. (2014).

Abstract modular inference systems and solvers.

In Flatt, M. and Guo, H., editors, *Proceedings of the 16th International Symposium on Practical Aspects of Declarative Languages , (PADL 2014)*, volume 8324 of *Lecture Notes in Computer Science*, pages 49–64. Springer.

Lierler, Y. and Truszczynski, M. (2015).

An abstract view on modularity in knowledge representation.

In Bonet, B. and Koenig, S., editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1532–1538. AAAI Press.

References XII

Marques-Silva, J. P. and Sakallah, K. A. (1996).

Conflict analysis in search algorithms for propositional satisfiability.

In *Proceedings of IEEE Conference on Tools with Artificial Intelligence*, pages 467–469. IEEE Computer Society.

Mellarkod, V. S., Gelfond, M., and Zhang, Y. (2008).

Integrating answer set programming and constraint logic programming.

Annals of Mathematics and Artificial Intelligence, 53(1-4):251–287.

Nieuwenhuis, R., Oliveras, A., and Tinelli, C. (2006).

Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T).

Journal of the ACM, 53(6):937–977.

References XIII

Nofal, S., Atkinson, K., and Dunne, P. E. (2014).

Algorithms for decision problems in argument systems under preferred semantics.

Artificial Intelligence, 207:23–51.

Ostrowski, M. and Schaub, T. (2012).

ASP modulo CSP: The clingcon system.

Theory and Practice of Logic programming (TPLP), 12(4-5):485–503.

Prosser, P. (1993).

Hybrid algorithms for the constraint satisfaction problem.

Computational Intelligence, 9:268–299.

References XIV

Simons, P., Niemelä, I., and Soininen, T. (2002).
Extending and implementing the stable model semantics.
Artificial Intelligence, 138:181–234.

Ward, J. and Schlipf, J. (2004).
Answer set programming with clause learning.
In *Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, pages 302–313.

Zhang, L., Madigan, C. F., Moskewicz, M. W., and Malik, S. (2001).
Efficient conflict driven learning in a boolean satisfiability solver.
In *Proceedings ICCAD-01*, pages 279–285.

APPENDIX

Definition of (plain) CNF conversion of Comp(Π)

Comp(Π) consists, for every $a \in atoms(\Pi)$ of clauses of two kinds:

1. the rules $a \leftarrow B$ of Π written as clauses

$$a \vee \overline{B}$$

2. formulas

$$\overline{a} \vee \bigvee_{B \in Bodies(\Pi,a)} B$$

converted to CNF using the distributivity of disjunction over conjunction (repetitions not removed).

Unfounded set: Definition

A set U of atoms occurring in a program Π is said to be unfounded on a consistent set M of literals w.r.t. Π if for every $a \in U$ and every $B \in \mathit{Bodies}(\Pi, a)$, at least one of the following conditions holds:

1. $\overline{B} \cap M \neq \emptyset$, or
2. $U \cap B+ \neq \emptyset$.

Loop formula: Definition

For any set Y of atoms, the external support formula for Y is

$$\vee_{B \in Bodies(\Pi, Y)} B$$

We denote the external support formula by $ES_{\Pi, Y}$. For any set Y of atoms, the loop formula for Y is the implication.

For any set Y of atoms, the loop formula for Y is the implication

$$\vee_{a \in Y} a \rightarrow ES_{\Pi, Y}.$$

Program entails a formula

We say that a program Π entails a formula F when for any consistent and complete set L of literals, if L^+ is an answer set for Π, then $L \models F$.

For instance, any program Π entails each rule occurring in Π.

SML_{Π}: Extended rules (I)

Conclude : $L \| \Gamma \Longrightarrow \mathit{UNSAT}$ if $\begin{cases} L \text{ is inconsistent and} \\ L \text{ contains no decision literals} \end{cases}$

Decide : $L \| \Gamma \Longrightarrow L l^{\Delta} \| \Gamma$ if $\begin{cases} L \text{ is consistent and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } L \end{cases}$

Unfounded : $L \| \Gamma \Longrightarrow L \bar{a} \| \Gamma$ if $\begin{cases} L \text{ is consistent and} \\ a \in U \text{ for a set } U \text{ unfounded on } L \text{ w.r.t. } \Pi \end{cases}$

SML_{Π}: Extended and updated rules (II)

UnitPropagateLP :
$$L\|\Gamma \Longrightarrow La\|\Gamma \quad \text{if} \begin{cases} \text{there is a rule } a \leftarrow B \text{ of } \Pi \text{ such that} \\ \quad B \subseteq L \end{cases}$$

AllRulesCancelled :
$$L\|\Gamma \Longrightarrow L\bar{a}\|\Gamma \quad \text{if} \begin{cases} \text{for each rule } a \leftarrow B \text{ of } \Pi \\ \quad B \text{ is contradicted by } L \end{cases}$$

BackchainTrue :
$$L\|\Gamma \Longrightarrow Ll\|\Gamma \quad \text{if} \begin{cases} \text{there is a rule } a \leftarrow l, B \text{ of } \Pi \text{ such that} \\ \quad a \text{ is in } L \text{ and} \\ \text{for each other rule } a \leftarrow B' \text{ of } \Pi \\ \quad B' \text{ is contradicted by } L \end{cases}$$

BackchainFalse :
$$L\|\Gamma \Longrightarrow L\bar{l}\|\Gamma \quad \text{if} \begin{cases} \text{there is a rule } a \leftarrow l, B \text{ of } \Pi \cup \Gamma \text{ s.t.} \\ \quad \bar{a} \text{ is in } L \text{ or } a = \bot \text{ and} \\ \quad B \subseteq L \end{cases}$$

SML_{Π}: Additional rules and *Success*

BackjumpLP : $Ll^{\Delta}L' \| \Gamma \Longrightarrow Ll' \| \Gamma$ if $\begin{cases} Ll^{\Delta}L' \text{ is inconsistent and} \\ \Pi \text{ entails } l' \vee \overline{L} \end{cases}$

LearnLP : $L \| \Gamma \Longrightarrow L \| \{\leftarrow B\} \cup \Gamma$ if $\{ \ \Pi \text{ entails } \overline{B}$

Success : $L \| \Gamma \Longrightarrow \ Ok(L)$ if no other rule applies other then *LearnLP*

SML_Π: Formal result

Theorem

For any program Π*,*

1. *every path in* SML_Π *uses only finitely many times edges justified by transition rules other than LearnLP,*
2. *any terminal state reachable from* $\emptyset||\emptyset$ *in* SML_Π *other than UNSAT is* $Ok(L)$*, with* L^+ *being an answer set of* Π*, and*
3. *UNSAT is reachable from* $\emptyset||\emptyset$ *in* SML_Π *if and only if* Π *has no answer sets.*

SML_{Π}: Example

Let Π be the following program:

$$a \leftarrow not\ b.$$
$$b \leftarrow not\ a.$$
$$c \leftarrow a.$$
$$d \leftarrow\ d.$$

Initial state :		$\emptyset \| \emptyset$
Decide	$\Longrightarrow$	$a^{\Delta} \| \emptyset$
UnitPropagateLP	$\Longrightarrow$	$a^{\Delta} c \| \emptyset$
AllRulesCancelled	$\Longrightarrow$	$a^{\Delta} c \overline{b} \| \emptyset$
Decide	$\Longrightarrow$	$a^{\Delta} c \overline{b} d^{\Delta} \| \emptyset$
Unfounded	$\Longrightarrow$	$a^{\Delta} c \overline{b} d^{\Delta} \overline{d} \| \emptyset$
BackjumpLP	$\Longrightarrow$	$a^{\Delta} c \overline{b}\ \overline{d} \| \emptyset$
LearnLP	$\Longrightarrow$	$a^{\Delta} c \overline{b}\ \overline{d} \| \{\overline{a} \vee \overline{c} \vee b \vee \overline{d}\}$
Success	$\Longrightarrow$	$Ok(a^{\Delta} c \overline{b}\ \overline{d})$

SML_{Π}: More rules

$$\textit{RestartLP}: \quad L||\Gamma \Longrightarrow \emptyset||\Gamma$$

$$\textit{ForgetLP}: \quad L||\{\leftarrow B\} \cup \Gamma \Longrightarrow L||\Gamma$$

Abstract SMODELS$_{cc}$

Abstract SMODELS$_{cc}$

SMODELS$_{cc}$ [Ward and Schlipf, 2004] can be abstractly described via SML_{Π}, and

1. assigns priorities to the application of the transition rules as follows:
   BackjumpLP, *Conclude* $>>$ *UnitPropagateLP*, *AllRulesCancelled*, *BackchainTrue*, *BackchainFalse*$>>$ *Unfounded* $>>$ *Decide*.
2. applies *LearnLP* in a state reached by the application of *BackjumpLP*.

Abstract SMODELS$_{cc}$: Example

Let Π be the following program:

$$a \leftarrow not\ b.$$
$$b \leftarrow not\ a.$$
$$c \leftarrow a.$$
$$d \leftarrow\ d.$$

Initial state :		$\emptyset \| \emptyset$
Decide	$\Longrightarrow$	$a^{\Delta} \| \emptyset$
UnitPropagateLP	$\Longrightarrow$	$a^{\Delta} c \| \emptyset$
AllRulesCancelled	$\Longrightarrow$	$a^{\Delta} c\overline{b} \| \emptyset$
Unfounded	$\Longrightarrow$	$a^{\Delta} c\overline{bd} \| \emptyset$
Success	$\Longrightarrow$	$Ok(a^{\Delta} c\overline{b}\ \overline{d})$

Figure : Example of path in SML_{Π} followed by abstract SMODELS$_{cc}$.