Systems and Solving Techniques for Knowledge Representation

– (Normal) ASP solving [Part I - SAT] –

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 part of the lectures the goal is to show how the main solvers for normal (non-disjunctive) ASP solvers, e.g.

- CMODELS,
- SMODELS,
- CLASP,
- ...

try to solved the program at hand.

We will NOT do these by presenting algorithm's behavior as usually done, i.e. by means of pseudo-code descriptions.

Algorithms presentation: Motivation for alternative way

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.

Algorithms presentation: Motivation for alternative way

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 $\perp$, 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 $\perp$, 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 $\perp$, 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 $\perp$, 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 $DPLL_F$

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

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

DPLL SAT solving: Example (I)

$\emptyset$

Figure : Example of path in $DPLL_{\{a \lor b, \bar{a} \lor c\}}$.

DPLL SAT solving: Example (I)

$$\emptyset \xrightarrow{D} a^{\Delta}$$

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

DPLL SAT solving: Example (I)

$$\emptyset \xrightarrow{D} a^{\Delta} \xrightarrow{U} a^{\Delta} c$$

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

DPLL SAT solving: Example (I)

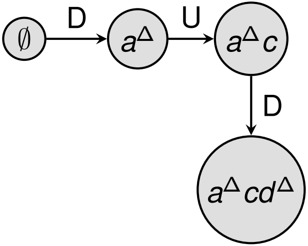

Figure : Example of path in $DPLL_{\{a \vee b, \bar{a} \vee c\}}$.

DPLL SAT solving: Example (I)

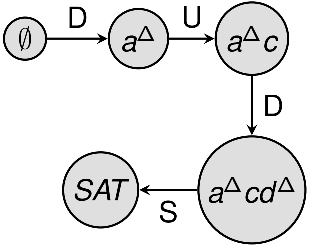

Figure : Example of path in $DPLL_{\{a \vee b, \bar{a} \vee c\}}$.

DPLL SAT solving: Another example (II)

$$\emptyset$$

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

DPLL SAT solving: Another xample (II)

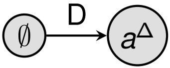

Figure : Example of path in $DPLL_{\{a \vee b, \bar{a} \vee c\}}$.

DPLL SAT solving: Another example (II)

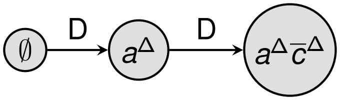

Figure : Example of path in $DPLL_{\{a \vee b, \overline{a} \vee c\}}$.

DPLL SAT solving: Another example (II)

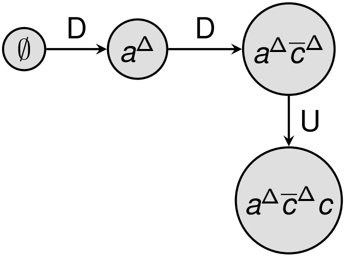

Figure : Example of path in $DPLL_{\{a \vee b, \overline{a} \vee c\}}$.

DPLL SAT solving: Another example (II)

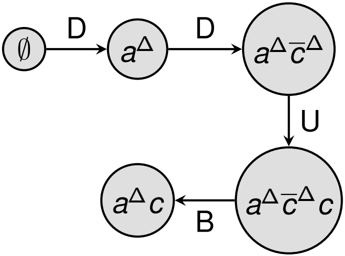

Figure : Example of path in $DPLL_{\{a \vee b, \overline{a} \vee c\}}$.

DPLL SAT solving: Another example (II)

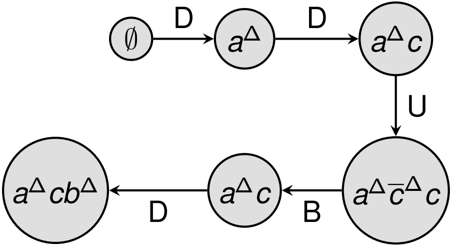

Figure : Example of path in $DPLL_{\{a\vee b, \overline{a}\vee c\}}$.

DPLL SAT solving: Another example (II)

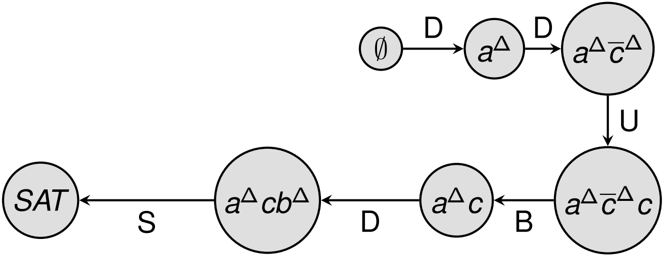

Figure : Example of path in $DPLL_{\{a \vee b, \overline{a} \vee c\}}$.

(Abstract) Solver description

Graph + Formula/Program's instantiation + Rule's ordering

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: Graphical alternative for 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 algorithm for SAT: Intuition

- Backjumping is the ability of, instead of backtracking chronologically to the last decision as in Backtrack, back-jumping over decision literals that “were not directly responsible for the inconsistency”;
- Learning adds clauses, to be conjoined with the original formula, in order to prevent that a similar inconsistency is encoutered again in another part of the search tree.

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 : $L \| \Gamma \Longrightarrow UNSAT$ 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

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

CDCL SAT solving: Updated and extended rules (II)

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

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

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

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

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 $DPLLearn_{\{a \vee b,\ \overline{a} \vee c\}}$.

CDCL SAT solving: Formal result

Theorem

For any formula F,

1. *every path in $DPLLearn_F$ uses only finitely many times edges justified by transition rules other than Learn,*
2. *any terminal state reachable from $\emptyset||\emptyset$ in $DPLLearn_F$ other than $UNSAT$ is SAT, and*
3. *$UNSAT$ is reachable from $\emptyset||\emptyset$ in $DPLLearn_F$ if and only if F is unsatisfiable.*

CDCL SAT solving: Additional transition rules (II)

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

When Learning comes into play, SAT solvers usually implement two more techniques

- Restart starts the search from scratch but mantaining the learned clauses;
- Forget deletes a previously added clause.

CDCL SAT solving: Additional transition rules (II)

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

$$\mathit{Restart}: \quad L||\Gamma \Longrightarrow \emptyset||\Gamma$$

$$\mathit{Forget}: \quad L||C \cup \Gamma \Longrightarrow L||\Gamma$$

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 : $L \Longrightarrow UNSAT$ 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 : $L \Longrightarrow Ok(L)$ 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

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

where

- the *head* a is an atom or $\bot$, 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

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

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

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

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 lecture, 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 lecture, 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 $GT_{F,G}$.

References I

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.

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.

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.

References II

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.

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

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.

References III

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.

Prosser, P. (1993).
Hybrid algorithms for the constraint satisfaction problem.
Computational Intelligence, 9:268–299.

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.