Satisfiability Modulo Theories
Lezione 3 - Efficient SAT-Solvers

(slides revision: Saturday 14$^{\text{th}}$ March, 2015, 11:46)

Roberto Bruttomesso

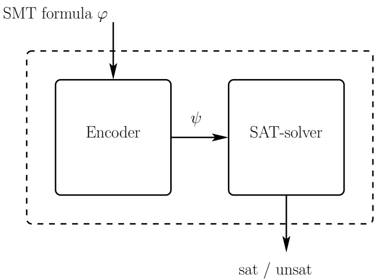
Seminario di Logica Matematica
(Corso Prof. Silvio Ghilardi)

3 Novembre 2011

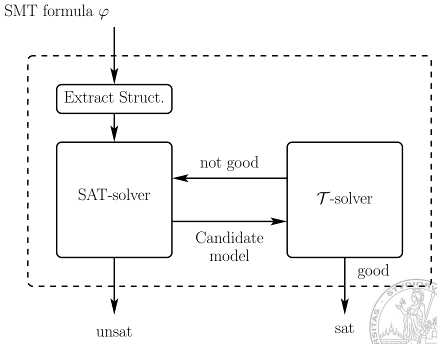
Recall from last lecture . . .

Approaches to solve SMT formulæ are based on the observation that SMT can be **reduced** to SAT, i.e., the purely Boolean Satisfiability Problem

Eager



Lazy



Outline

Disclaimer

The SAT-Solving algorithm described as follows is **not** the only approach existing

- BDDs
- Quantifier Elimination
- Random SAT-Solving

We study the DPLL/CDCL approach as it is precise (not approximate), it uses a linear amount of memory, and very robust

It is fair to say that it is the most used approach by industry for pure solving

The approach evolved in many years, many groups have contributed. Here we do not see the history of this evolution but just the **final product**

Complexity Considerations

SAT is "the" NP-Complete problem

It is unlikely to be solved in polynomial time

Most likely, it takes some $O(2^n)$ time complexity for an algorithm to solve SAT

Complexity of SAT cannot be alleviated by faster machines (only by parallelism, if we ever reach that technology). Suppose you can execute M instructions in 1 hour, then the maximum n you can handle is

Speed of machine	Max input size in 1 hour
1x	$log_2(M) = n$
100x	$log_2(100 \cdot M) \approx n + 3.3$
1000000x	$log_2(1000000 \cdot M) \approx n + 19$

CNF Formulæ

From now on we shall focus on solving formulæ in Conjunctive Normal Form **CNF**

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

where each C_i is a **clause**, a disjunction of the kind

$$(l_1 \vee l_2 \vee \ldots l_m)$$

where every l_i is a **literal**, which is a variable or a negated Boolean variable

$$a \quad \text{or} \quad \neg a$$

CNF Formulæ

From now on we shall focus on solving formulæ in Conjunctive Normal Form **CNF**

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

where each C_i is a **clause**, a disjunction of the kind

$$(l_1 \vee l_2 \vee \ldots l_m)$$

where every l_i is a **literal**, which is a variable or a negated Boolean variable

$$a \quad \text{or} \quad \neg a$$

For simplicity we will write them as a set of clauses, omitting the $\wedge$, like

$$\begin{array}{l} (\neg a_1 \vee a_2) \\ (\neg a_1 \vee a_3 \vee a_9) \\ (\neg a_2 \vee \neg a_3 \vee a_4) \\ (\neg a_4 \vee a_5 \vee a_{10}) \end{array}$$

Basic Notation

We indicate a (possibly partial) **assignment** as the set of literals that are $\top$ under it, i.e., we write the assignment

$$\{a_1 \mapsto \top, a_2 \mapsto \top, a_3 \mapsto \bot, a_4 \mapsto \bot, a_9 \mapsto \top\}$$

as

$$\{a_1, a_2, \neg a_3, \neg a_4, a_9\}$$

Basic Notation

We indicate a (possibly partial) **assignment** as the set of literals that are $\top$ under it, i.e., we write the assignment

$$\{a_1 \mapsto \top, a_2 \mapsto \top, a_3 \mapsto \bot, a_4 \mapsto \bot, a_9 \mapsto \top\}$$

as

$$\{a_1, a_2, \neg a_3, \neg a_4, a_9\}$$

An assignment is used to evaluate a formula, e.g.,

$$\begin{array}{l} (\neg a_1 \vee a_2) \\ (\neg a_1 \vee a_3 \vee a_9) \\ (\neg a_2 \vee \neg a_3 \vee a_4) \\ (\neg a_4 \vee a_5 \vee a_{10}) \end{array}$$

evaluates to $\top$ under the assignment above

Basic Notation

We indicate a (possibly partial) **assignment** as the set of literals that are $\top$ under it, i.e., we write the assignment

$$\{a_1 \mapsto \top, a_2 \mapsto \top, a_3 \mapsto \bot, a_4 \mapsto \bot, a_9 \mapsto \top\}$$

as

$$\{a_1, a_2, \neg a_3, \neg a_4, a_9\}$$

An assignment is used to evaluate a formula, e.g.,

$$\begin{array}{l} (\neg a_1 \vee a_2) \\ (\neg a_1 \vee a_3 \vee a_9) \\ (\neg a_2 \vee \neg a_3 \vee a_4) \\ (\neg a_4 \vee a_5 \vee a_{10}) \end{array}$$

evaluates to $\top$ under the assignment above

- at least one this colored literal in each clause to make it $\top$
- all this colored literals in one clause to make it $\bot$

Simple Facts

SAT-Solving is (the art of) finding the assignment satisfying all clauses

Simple Facts

SAT-Solving is (the art of) finding the assignment satisfying all clauses

The assignment evolves **incrementally** by taking **Decisions**, starting from empty { }, but it can be **backtracked** if found wrong

Simple Facts

SAT-Solving is (the art of) finding the assignment satisfying all clauses

The assignment evolves **incrementally** by taking **Decisions**, starting from empty $\{\ \}$, but it can be **backtracked** if found wrong

The evolution of the assignment can be represented as a **tree**

Simple Facts

SAT-Solving is (the art of) finding the assignment satisfying all clauses

The assignment evolves **incrementally** by taking **Decisions**, starting from empty { }, but it can be **backtracked** if found wrong

The evolution of the assignment can be represented as a **tree**

$$(\neg a_1 \vee a_3 \vee a_9)$$
$$(\neg a_2 \vee \neg a_3 \vee a_4)$$
$$(a_1 \vee a_2)$$
$$(\neg a_4 \vee a_5 \vee a_{10})$$

{ }

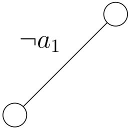
Simple Facts

SAT-Solving is (the art of) finding the assignment satisfying all clauses

The assignment evolves **incrementally** by taking **Decisions**, starting from empty { }, but it can be **backtracked** if found wrong

The evolution of the assignment can be represented as a **tree**

$$(\neg a_1 \vee a_3 \vee a_9)$$
$$(\neg a_2 \vee \neg a_3 \vee a_4)$$
$$(a_1 \vee a_2)$$
$$(\neg a_4 \vee a_5 \vee a_{10})$$



$\{\neg a_1\}$

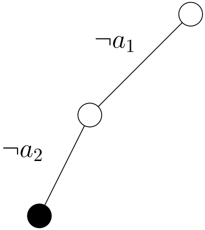
SAT-Solving is (the art of) finding the assignment satisfying all clauses

The assignment evolves **incrementally** by taking **Decisions**, starting from empty { }, but it can be **backtracked** if found wrong

The evolution of the assignment can be represented as a **tree**

$$(\neg a_1 \vee a_3 \vee a_9)$$
$$(\neg a_2 \vee \neg a_3 \vee a_4)$$
$$(a_1 \vee a_2)$$
$$(\neg a_4 \vee a_5 \vee a_{10})$$

$\{\neg a_1, \neg a_2\}$



Simple Facts

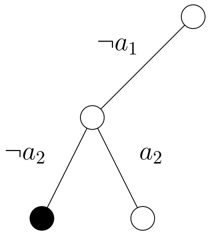
SAT-Solving is (the art of) finding the assignment satisfying all clauses

The assignment evolves **incrementally** by taking **Decisions**, starting from empty { }, but it can be **backtracked** if found wrong

The evolution of the assignment can be represented as a **tree**

$$(\neg a_1 \vee a_3 \vee a_9)$$
$$(\neg a_2 \vee \neg a_3 \vee a_4)$$
$$(a_1 \vee a_2)$$
$$(\neg a_4 \vee a_5 \vee a_{10})$$

$\{\neg a_1, a_2\}$



Simple Facts

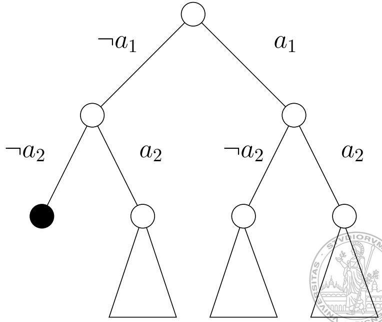
SAT-Solving is (the art of) finding the assignment satisfying all clauses

The assignment evolves **incrementally** by taking **Decisions**, starting from empty { }, but it can be backtracked if found wrong

The evolution of the assignment can be represented as a **tree**

$$(\neg a_1 \vee a_3 \vee a_9)$$
$$(\neg a_2 \vee \neg a_3 \vee a_4)$$
$$(a_1 \vee a_2)$$
$$(\neg a_4 \vee a_5 \vee a_{10})$$

$\{\ldots\}$



Simple Facts

There are assignments which can be trivially driven towards the right direction. In the example below, given the current assignment, the third clause can be satisfied only by setting $a_2 \mapsto \top$

$$(\neg a_1 \vee a_3 \vee a_9)$$
$$(\neg a_2 \vee \neg a_3 \vee a_4)$$
$$(a_1 \vee a_2)$$
$$(\neg a_4 \vee a_5 \vee a_{10})$$

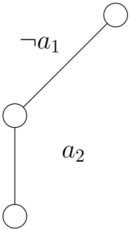
$\{\neg a_1\}$

Simple Facts

There are assignments which can be trivially driven towards the right direction. In the example below, given the current assignment, the third clause can be satisfied only by setting $a_2 \mapsto \top$

$$(\neg a_1 \vee a_3 \vee a_9)$$
$$(\neg a_2 \vee \neg a_3 \vee a_4)$$
$$(a_1 \vee a_2)$$
$$(\neg a_4 \vee a_5 \vee a_{10})$$

$\{\neg a_1, a_2\}$

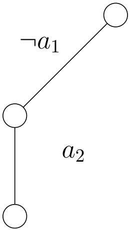


Simple Facts

There are assignments which can be trivially driven towards the right direction. In the example below, given the current assignment, the third clause can be satisfied only by setting $a_2 \mapsto \top$

$$
\begin{aligned}
&(\neg a_1 \vee a_3 \vee a_9)\\
&(\neg a_2 \vee \neg a_3 \vee a_4)\\
&(a_1 \vee a_2)\\
&(\neg a_4 \vee a_5 \vee a_{10})
\end{aligned}
$$

$\{\neg a_1, a_2\}$



This step is called **Boolean Constraint Propagation** (BCP). It represents a **forced** deduction. It triggers whenever all literals but one are assigned $\perp$

$$(\neg a_1 \vee a_2 \vee \neg a_3 \vee a_4 \vee \neg a_5)$$

Simple Facts

Decision-level: in an assignment, it is the number of decisions taken

Clearly, BCPs do not contribute to increase the decision level

When necessary, we will indicate decision level on top of literals $\overset{5}{a}$ in the assignment

Simple Facts

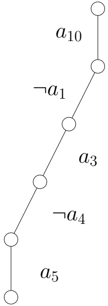
Decision-level: in an assignment, it is the number of decisions taken

Clearly, BCPs do not contribute to increase the decision level

When necessary, we will indicate decision level on top of literals $\overset{5}{a}$ in the assignment

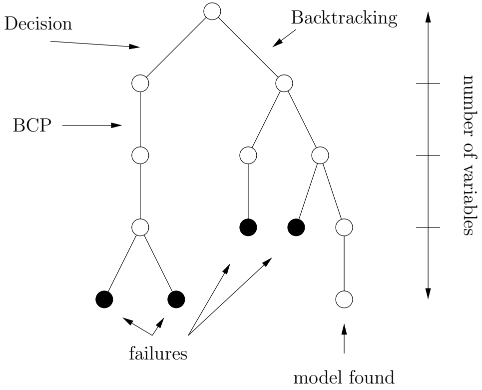
Example:

$$\{\overset{0}{a_{10}}, \overset{1}{\neg a_1}, \overset{2}{a_3}, \overset{3}{\neg a_4}, \overset{3}{a_5}\}$$



Simple Facts

The process of finding the satisfying assignment is called **search**, and in its basic version it evolves with **Decisions**, **BCPs**, and **backtracking**



The DPLL Procedure (Revised)

DPLL(V, $\mathcal{C}$)

if (“a clause has all $\perp$ literals”) return *unsat*; // Failure
if (“all clauses has a $\top$ literal”) return *sat*; // Model found

The DPLL Procedure (Revised)

DPLL(V, $\mathcal{C}$)

if ("a clause has all $\bot$ literals") return *unsat*; // Failure
if ("all clauses has a $\top$ literal") return *sat*; // Model found

if ("a clause has all but one literal l to $\bot$")
return DPLL($V \cup \{l\}$, $\mathcal{C}$); // BCP

The DPLL Procedure (Revised)

DPLL(V, $\mathcal{C}$)

```
if ( "a clause has all ⊥ literals" ) return unsat;  // Failure
if ( "all clauses has a ⊤ literal" ) return sat;    // Model found

if ( "a clause has all but one literal l to ⊥" )
  return DPLL( V ∪ {l}, C );                        // BCP

l = "pick an unassigned literal"                    // Decision
```

The DPLL Procedure (Revised)

DPLL(V, $\mathcal{C}$)

  if (“a clause has all $\perp$ literals”) return *unsat*; // Failure
  if (“all clauses has a $\top$ literal”) return *sat*; // Model found

  if (“a clause has all but one literal l to $\perp$”)
    return DPLL($V \cup \{l\}$, $\mathcal{C}$); // BCP

  l = “pick an unassigned literal” // Decision

  if (DPLL($V \cup \{l\}$, $\mathcal{C}$) == *sat*) // Left Branch
    return *sat*;
  else
    return DPLL($V \cup \{\neg l\}$, $\mathcal{C}$); // Backtracking (Rigth Branch)

The DPLL Procedure (Revised)

DPLL(V, $\mathcal{C}$)

```
  if ( "a clause has all ⊥ literals" ) return unsat;  // Failure
  if ( "all clauses has a ⊤ literal" ) return sat;    // Model found

  if ( "a clause has all but one literal l to ⊥" )
    return DPLL( V ∪ {l}, C );                        // BCP

  l = "pick an unassigned literal"                    // Decision

  if ( DPLL( V ∪ {l}, C ) == sat )                    // Left Branch
    return sat;
  else
    return DPLL( V ∪ {¬l}, C );                       // Backtracking (Rigth Branch)
```

To be called as DPLL({ }, $\mathcal{C}$)

The Iterative DPLL Procedure (Revised)

```
dl = 0; flipped = { }; trail = { };                    // Decision level, flipped vars, assignment

while ( true )

  if ( BCP( ) == conflict )                            // Do BCP until possible
    done = false;
    do
      if ( dl == 0 ) return unsat;                     // Unresolvable conflict
      l = GetDecisionLiteralAt( dl );
      BacktrackTo( dl − 1 );                           // Backtracking (shrinks trail)
      if ( var(l) ∈ flipped )
         dl = dl − 1;
      else
         trail = trail ∪ {¬l};
         flipped = flipped ∪ {var(l)};
         done = true;
    while ( !done );

  else
    if ( "all variables assigned" ) return sat;        // trail holds satisfying assignment
    l = Decision( );                                   // Do another decision
    trail = trail ∪ {l}
    dl = dl + 1;                                       // Increase decision level
```

The DPLL Procedure (Revised)

Some characteristics

- BCP is called with highest priority, as much as possible
- Backtracking is performed **chronologically**: search backtracks to the previous decision-level

The DPLL Procedure (Revised)

Some characteristics

- BCP is called with highest priority, as much as possible
- Backtracking is performed **chronologically**: search backtracks to the previous decision-level

Some differences w.r.t. "original" DPLL Procedure

- $\mathcal{C}$ is not touched: non-destructive data-structures (memory efficient)
- Pure Literal Rule is not used: expensive to implement (!)

From DPLL to CDCL

Consider the following scenario before and after BCP

$\ldots$
$(\neg a_{10} \vee \neg a_1 \vee a_4)$
$(a_3 \vee \neg a_1 \vee a_5)$
$(\neg a_4 \vee a_6)$
$(\neg a_5 \vee \neg a_6)$
$\ldots$

$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\}$

$\ldots$
$(\neg a_{10} \vee \neg a_1 \vee a_4)$
$(a_3 \vee \neg a_1 \vee a_5)$
$(\neg a_4 \vee a_6)$
$(\neg a_5 \vee \neg a_6)$
$\ldots$

$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$

From DPLL to CDCL

Consider the following scenario before and after BCP

$$\begin{array}{l} \dots \\ (\neg a_{10} \vee \neg a_1 \vee a_4) \\ (a_3 \vee \neg a_1 \vee a_5) \\ (\neg a_4 \vee a_6) \\ (\neg a_5 \vee \neg a_6) \\ \dots \end{array} \qquad \begin{array}{l} \dots \\ (\neg a_{10} \vee \neg a_1 \vee a_4) \\ (a_3 \vee \neg a_1 \vee a_5) \\ (\neg a_4 \vee a_6) \\ (\neg a_5 \vee \neg a_6) \\ \dots \end{array}$$

$$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\} \qquad \{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$$

BCP leads to a conflict, but, what is the **reason** for it ? Do a_7 and a_2 play any role ?

From DPLL to CDCL

Consider the following scenario before and after BCP

$$\begin{array}{ll}
\ldots & \ldots \\
(\neg a_{10} \vee \neg a_1 \vee a_4) & (\neg a_{10} \vee \neg a_1 \vee a_4) \\
(a_3 \vee \neg a_1 \vee a_5) & (a_3 \vee \neg a_1 \vee a_5) \\
(\neg a_4 \vee a_6) & (\neg a_4 \vee a_6) \\
(\neg a_5 \vee \neg a_6) & (\neg a_5 \vee \neg a_6) \\
\ldots & \ldots \\
\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\} & \{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}
\end{array}$$

BCP leads to a conflict, but, what is the **reason** for it ? Do a_7 and a_2 play any role ?

Does it make sense to consider the assignments (which backtracking would produce) ?

$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{\neg a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\}$ — $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\}$ — $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{a_2}, \overset{4}{a_1}\}$

From DPLL to CDCL

Consider the following scenario before and after BCP

$\ldots$	$\ldots$
$(\neg a_{10} \vee \neg a_1 \vee a_4)$	$(\neg a_{10} \vee \neg a_1 \vee a_4)$
$(a_3 \vee \neg a_1 \vee a_5)$	$(a_3 \vee \neg a_1 \vee a_5)$
$(\neg a_4 \vee a_6)$	$(\neg a_4 \vee a_6)$
$(\neg a_5 \vee \neg a_6)$	$(\neg a_5 \vee \neg a_6)$
$\ldots$	$\ldots$
$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\}$	$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$

BCP leads to a conflict, but, what is the **reason** for it ? Do a_7 and a_2 play any role ?

Does it make sense to consider the assignments (which backtracking would produce) ?
$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{\neg a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\}$ — $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\}$ — $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{a_2}, \overset{4}{a_1}\}$

No, because *whenever* $a_{10}, \neg a_3$ *is assigned, then* a_1 *must not be set to* $\top$.

From DPLL to CDCL

Consider the following scenario before and after BCP

$\ldots$	$\ldots$
$(\neg a_{10} \vee \neg a_1 \vee a_4)$	$(\neg a_{10} \vee \neg a_1 \vee a_4)$
$(a_3 \vee \neg a_1 \vee a_5)$	$(a_3 \vee \neg a_1 \vee a_5)$
$(\neg a_4 \vee a_6)$	$(\neg a_4 \vee a_6)$
$(\neg a_5 \vee \neg a_6)$	$(\neg a_5 \vee \neg a_6)$
$\ldots$	$\ldots$
$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\}$	$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$

BCP leads to a conflict, but, what is the **reason** for it ? Do a_7 and a_2 play any role ?

Does it make sense to consider the assignments (which backtracking would produce) ?
$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{\neg a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\}$ — $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}\}$ — $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{a_2}, \overset{4}{a_1}\}$

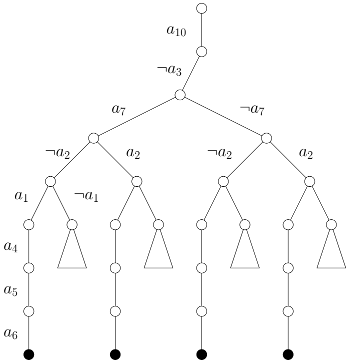
No, because *whenever* $a_{10}, \neg a_3$ *is assigned, then* a_1 *must not be set to* $\top$. This translates to an additional clause, which can be *learnt*, i.e., it can be added to the formula

$$(\neg a_{10} \vee a_3 \vee \neg a_1)$$

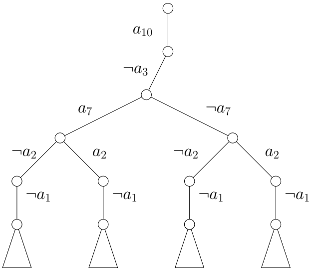
From DPLL to CDCL

Search

Without learnt clause



With learnt clause



Deriving the clause to be learnt

The clause to be learnt is derived from the conflict “caused” by BCP (conflict is never caused by a Decision)

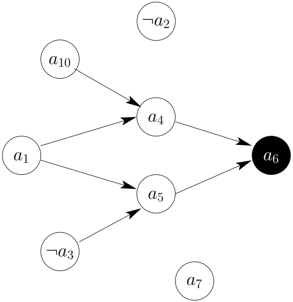
Deriving the clause to be learnt

The clause to be learnt is derived from the conflict "caused" by BCP (conflict is never caused by a Decision)

To better **analyze** a conflict we need to look at the **implication graph**

$\dots$
$(\neg a_{10} \vee \neg a_1 \vee a_4)$
$(a_3 \vee \neg a_1 \vee a_5)$
$(\neg a_4 \vee a_6)$
$(\neg a_5 \vee \neg a_6)$
$\dots$

$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$



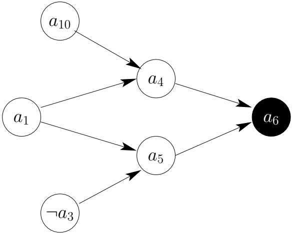
Deriving the clause to be learnt

The clause to be learnt is derived from the conflict "caused" by BCP (conflict is never caused by a Decision)

To better **analyze** a conflict we need to look at the **implication graph**

$\ldots$
$(\neg a_{10} \vee \neg a_1 \vee a_4)$
$(a_3 \vee \neg a_1 \vee a_5)$
$(\neg a_4 \vee a_6)$
$(\neg a_5 \vee \neg a_6)$
$\ldots$

$\{\overset{0}{a_{10}}, \neg \overset{1}{a_3}, \overset{2}{a_7}, \neg \overset{3}{a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$



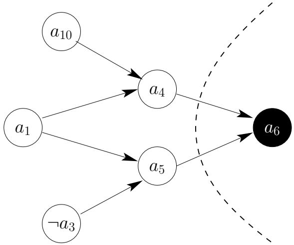
Deriving the clause to be learnt

The clause to be learnt is derived from the conflict "caused" by BCP (conflict is never caused by a Decision)

To better **analyze** a conflict we need to look at the **implication graph**

$$
\begin{array}{l}
\ldots \\
(\neg a_{10} \vee \neg a_1 \vee a_4) \\
(a_3 \vee \neg a_1 \vee a_5) \\
(\neg a_4 \vee a_6) \\
(\neg a_5 \vee \neg a_6) \\
\ldots
\end{array}
$$

$$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$$



Any cut of the graph that separates the conflict from the decision variables represents a possible learnt clause

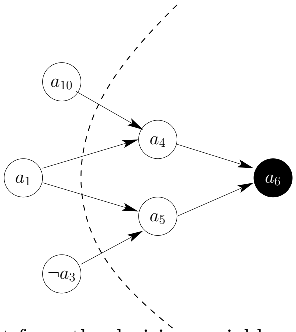
Deriving the clause to be learnt

The clause to be learnt is derived from the conflict "caused" by BCP (conflict is never caused by a Decision)

To better **analyze** a conflict we need to look at the **implication graph**

$\ldots$
$(\neg a_{10} \vee \neg a_1 \vee a_4)$
$(a_3 \vee \neg a_1 \vee a_5)$
$(\neg a_4 \vee a_6)$
$(\neg a_5 \vee \neg a_6)$
$\ldots$

$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$



Any cut of the graph that separates the conflict from the decision variables represents a possible learnt clause

In this course we take the clause that contains **only one variable of the current decision level**

Deriving the clause to be learnt

In practice, we use **resolution** steps to compute the learnt clause:

- we start from the clause with all literals to $\perp$ (conflicting clause)
- iteratively, we take the clause that propagated the last literal on the trail and we apply resolution
- we stop when only one literal from the current decision level is left in the clause

Deriving the clause to be learnt

In practice, we use **resolution** steps to compute the learnt clause:

- we start from the clause with all literals to $\perp$ (conflicting clause)
- iteratively, we take the clause that propagated the last literal on the trail and we apply resolution
- we stop when only one literal from the current decision level is left in the clause

$(\neg a_5 \vee \neg a_6)$

Trail	dl	Reason
a_{10}	0	(a_{10})
$\neg a_3$	1	Decision
a_7	2	Decision
$\neg a_2$	3	Decision
a_1	4	Decision
a_4	4	$(\neg a_{10} \vee \neg a_1 \vee a_4)$
a_5	4	$(a_3 \vee \neg a_1 \vee a_5)$
a_6	4	$(\neg a_4 \vee a_6)$

$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$

Deriving the clause to be learnt

In practice, we use **resolution** steps to compute the learnt clause:

- we start from the clause with all literals to $\bot$ (conflicting clause)
- iteratively, we take the clause that propagated the last literal on the trail and we apply resolution
- we stop when only one literal from the current decision level is left in the clause

$(\neg a_5 \vee \neg a_6) \quad (\neg a_4 \vee a_6)$

Trail	dl	Reason
a_{10}	0	(a_{10})
$\neg a_3$	1	Decision
a_7	2	Decision
$\neg a_2$	3	Decision
a_1	4	Decision
a_4	4	$(\neg a_{10} \vee \neg a_1 \vee a_4)$
a_5	4	$(a_3 \vee \neg a_1 \vee a_5)$
a_6	4	$(\neg a_4 \vee a_6)$

$$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$$

Deriving the clause to be learnt

In practice, we use **resolution** steps to compute the learnt clause:

- we start from the clause with all literals to $\perp$ (conflicting clause)
- iteratively, we take the clause that propagated the last literal on the trail and we apply resolution
- we stop when only one literal from the current decision level is left in the clause

$$\frac{(\neg a_5 \vee \neg a_6) \quad (\neg a_4 \vee a_6)}{(\neg a_5 \vee \neg a_4)}$$

Trail	dl	Reason
a_{10}	0	(a_{10})
$\neg a_3$	1	Decision
a_7	2	Decision
$\neg a_2$	3	Decision
a_1	4	Decision
a_4	4	$(\neg a_{10} \vee \neg a_1 \vee a_4)$
a_5	4	$(a_3 \vee \neg a_1 \vee a_5)$
a_6	4	$(\neg a_4 \vee a_6)$

$$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$$

Deriving the clause to be learnt

In practice, we use **resolution** steps to compute the learnt clause:

- we start from the clause with all literals to $\perp$ (conflicting clause)
- iteratively, we take the clause that propagated the last literal on the trail and we apply resolution
- we stop when only one literal from the current decision level is left in the clause

$$\frac{(\neg a_5 \vee \neg a_6) \quad (\neg a_4 \vee a_6)}{(\neg a_5 \vee \neg a_4)} \quad (a_3 \vee \neg a_1 \vee a_5)$$

Trail	dl	Reason
a_{10}	0	(a_{10})
$\neg a_3$	1	Decision
a_7	2	Decision
$\neg a_2$	3	Decision
a_1	4	Decision
a_4	4	$(\neg a_{10} \vee \neg a_1 \vee a_4)$
a_5	4	$(a_3 \vee \neg a_1 \vee a_5)$
a_6	4	$(\neg a_4 \vee a_6)$

$$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$$

Deriving the clause to be learnt

In practice, we use **resolution** steps to compute the learnt clause:

- we start from the clause with all literals to $\bot$ (conflicting clause)
- iteratively, we take the clause that propagated the last literal on the trail and we apply resolution
- we stop when only one literal from the current decision level is left in the clause

$$\dfrac{\dfrac{(\neg a_5 \vee \neg a_6) \quad (\neg a_4 \vee a_6)}{(\neg a_5 \vee \neg a_4)} \quad (a_3 \vee \neg a_1 \vee a_5)}{(\neg a_4 \vee a_3 \vee \neg a_1)}$$

Trail	dl	Reason
a_{10}	0	(a_{10})
$\neg a_3$	1	Decision
a_7	2	Decision
$\neg a_2$	3	Decision
a_1	4	Decision
a_4	4	$(\neg a_{10} \vee \neg a_1 \vee a_4)$
a_5	4	$(a_3 \vee \neg a_1 \vee a_5)$
a_6	4	$(\neg a_4 \vee a_6)$

$$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$$

Deriving the clause to be learnt

In practice, we use **resolution** steps to compute the learnt clause:

- we start from the clause with all literals to $\perp$ (conflicting clause)
- iteratively, we take the clause that propagated the last literal on the trail and we apply resolution
- we stop when only one literal from the current decision level is left in the clause

$$\dfrac{\dfrac{(\neg a_5 \vee \neg a_6) \quad (\neg a_4 \vee a_6)}{(\neg a_5 \vee \neg a_4)} \quad (a_3 \vee \neg a_1 \vee a_5)}{(\neg a_4 \vee a_3 \vee \neg a_1)} \quad (\neg a_{10} \vee \neg a_1 \vee a_4)$$

Trail	dl	Reason
a_{10}	0	(a_{10})
$\neg a_3$	1	Decision
a_7	2	Decision
$\neg a_2$	3	Decision
a_1	4	Decision
a_4	4	$(\neg a_{10} \vee \neg a_1 \vee a_4)$
a_5	4	$(a_3 \vee \neg a_1 \vee a_5)$
a_6	4	$(\neg a_4 \vee a_6)$

$$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$$

Deriving the clause to be learnt

In practice, we use **resolution** steps to compute the learnt clause:

- we start from the clause with all literals to $\perp$ (conflicting clause)
- iteratively, we take the clause that propagated the last literal on the trail and we apply resolution
- we stop when only one literal from the current decision level is left in the clause

$$\cfrac{\cfrac{\cfrac{(\neg a_5 \vee \neg a_6) \quad (\neg a_4 \vee a_6)}{(\neg a_5 \vee \neg a_4)} \quad (a_3 \vee \neg a_1 \vee a_5)}{(\neg a_4 \vee a_3 \vee \neg a_1)} \quad (\neg a_{10} \vee \neg a_1 \vee a_4)}{(\neg a_{10} \vee a_3 \vee \neg a_1)}$$

Trail	dl	Reason
a_{10}	0	(a_{10})
$\neg a_3$	1	Decision
a_7	2	Decision
$\neg a_2$	3	Decision
a_1	4	Decision
a_4	4	$(\neg a_{10} \vee \neg a_1 \vee a_4)$
a_5	4	$(a_3 \vee \neg a_1 \vee a_5)$
a_6	4	$(\neg a_4 \vee a_6)$

$$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$$

Deriving the clause to be learnt

In practice, we use **resolution** steps to compute the learnt clause:

- we start from the clause with all literals to $\perp$ (conflicting clause)
- iteratively, we take the clause that propagated the last literal on the trail and we apply resolution
- we stop when only one literal from the current decision level is left in the clause

$$\cfrac{\cfrac{\cfrac{(\neg a_5 \vee \neg a_6) \quad (\neg a_4 \vee a_6)}{(\neg a_5 \vee \neg a_4) \quad (a_3 \vee \neg a_1 \vee a_5)}}{(\neg a_4 \vee a_3 \vee \neg a_1) \quad (\neg a_{10} \vee \neg a_1 \vee a_4)}}{(\neg a_{10} \vee a_3 \vee \neg a_1)}$$

Trail	dl	Reason
a_{10}	0	(a_{10})
$\neg a_3$	1	Decision
a_7	2	Decision
$\neg a_2$	3	Decision
a_1	4	Decision
a_4	4	$(\neg a_{10} \vee \neg a_1 \vee a_4)$
a_5	4	$(a_3 \vee \neg a_1 \vee a_5)$
a_6	4	$(\neg a_4 \vee a_6)$

$$\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$$

We say that $(\neg a_{10} \vee a_3 \vee \neg a_1)$ is the **conflict clause** and it is the one we learn

Non-chronological backtracking

So far we have been backtracking **chronologically**, but

given our (wrong) assignment $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$ and the computed conflict clause $(\neg a_{10} \vee a_3 \vee \neg a_1)$, is it really clever to backtrack to level 3 ?

Non-chronological backtracking

So far we have been backtracking **chronologically**, but

given our (wrong) assignment $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$ and the computed conflict clause $(\neg a_{10} \vee a_3 \vee \neg a_1)$, is it really clever to backtrack to level 3 ?

The correct level to backtrack to is:

> "the level that would have propagated $\neg a_1$ if we had the clause $(\neg a_{10} \vee a_3 \vee \neg a_1)$ as part of the original formula"

Non-chronological backtracking

So far we have been backtracking **chronologically**, but

given our (wrong) assignment $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$ and the computed conflict clause $(\neg a_{10} \vee a_3 \vee \neg a_1)$, is it really clever to backtrack to level 3 ?

The correct level to backtrack to is:

> "the level that would have propagated $\neg a_1$ if we had the clause $(\neg a_{10} \vee a_3 \vee \neg a_1)$ as part of the original formula"

In our case, it is level 1, because assignment $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}\}$ is sufficient to propagate $\neg a_1$ in $(\neg a_{10} \vee a_3 \vee \neg a_1)$

Non-chronological backtracking

So far we have been backtracking **chronologically**, but

given our (wrong) assignment $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}, \overset{2}{a_7}, \overset{3}{\neg a_2}, \overset{4}{a_1}, \overset{4}{a_4}, \overset{4}{a_5}, \overset{4}{a_6}\}$ and the computed conflict clause $(\neg a_{10} \vee a_3 \vee \neg a_1)$, is it really clever to backtrack to level 3 ?

The correct level to backtrack to is:

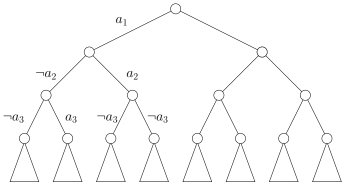
> "the level that would have propagated $\neg a_1$ if we had the clause $(\neg a_{10} \vee a_3 \vee \neg a_1)$ as part of the original formula"

In our case, it is level 1, because assignment $\{\overset{0}{a_{10}}, \overset{1}{\neg a_3}\}$ is sufficient to propagate $\neg a_1$ in $(\neg a_{10} \vee a_3 \vee \neg a_1)$

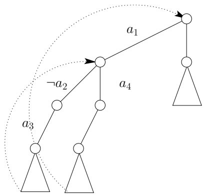
We say that $\neg a_1$ is the **asserting literal** as it becomes true by BCP once we have backtracked to the correct decision level

Search

DPLL
no learning
chronological backtracking



CDCL
conflict-driven learning
non-chronological backtracking



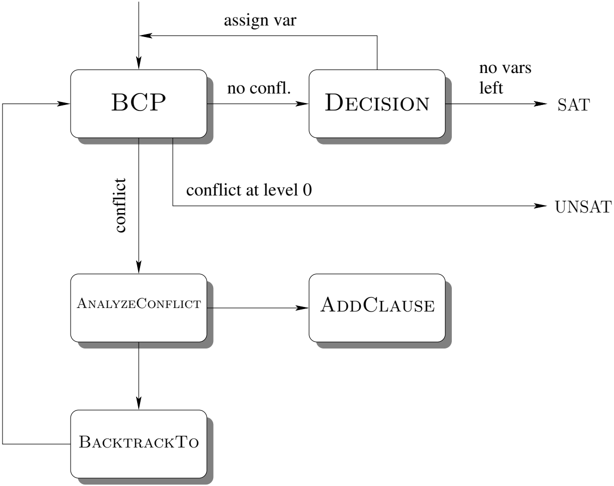
The CDCL Procedure

```
dl = 0; trail = { };                                   // Decision level, assignment

while ( true )

  if ( BCP( ) == conflict )                            // Do BCP until possible
    if ( dl == 0 ) return unsat;                       // Unresolvable conflict
    C, dl = AnalyzeConflict( );                        // Compute conf. clause, and dec. level
    AddClause( C );                                    // Add C to clause database
    BacktrackTo( dl );                                 // Backtracking (shrinks trail)
  else
    if ( "all variables assigned" ) return sat;        // trail holds satisfying assignment
    l = Decision( );                                   // Do another decision
    trail = trail ∪ {l}
    dl = dl + 1;                                       // Increase decision level
```

The CDCL Procedure



Other things which we did not see

Watched Literals: technique to efficiently discover which clauses become unsat

Decision heuristics: how do we choose the right variable ? And which polarity ? $(\top, \bot)$. Landmark strategy is VSIDS heuristic

Clause removal: adding too many clauses negatively impacts performance, need mechanisms to remove learnts

Restarts: start the search from scratch, but retaining learnts

many many other heuristics discovered every year

Exercise

Consider the following clause set and assignment

$$
\begin{array}{l}
(\neg a_1 \vee a_2) \\
(\neg a_1 \vee a_3 \vee a_9) \\
(\neg a_2 \vee \neg a_3 \vee a_4) \\
(\neg a_4 \vee a_5 \vee a_{10}) \\
(\neg a_4 \vee a_6 \vee a_{11}) \\
(\neg a_5 \vee \neg a_6) \\
(a_1 \vee a_7 \vee \neg a_{12}) \\
(a_1 \vee a_8) \\
(\neg a_7 \vee \neg a_8 \vee \neg a_{13}) \\
\ldots
\end{array}
$$

$$\{\overset{1}{\neg a_9}, \overset{2}{a_{12}}, \overset{2}{a_{13}}, \overset{3}{\neg a_{10}}, \overset{3}{\neg a_{11}}, \ldots, \overset{6}{a_1}\}$$

- Find the correct conflict clause
- Find the correct decision level to backtrack