Lecture Notes for

Logic and Computability

Course Number: IND04033UF

Contact

Bettina Könighofer
Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Austria
bettina.koenighofer@iaik.tugraz.at

2 SAT Solvers

Determining the satisfiability of a formula stands as a powerful technique, given that numerous real-world problems can be framed as instances of the SAT problem. Examples lie within the areas of high-level planning, scheduling, artificial intelligence, circuit testing, and software verification. The practical importance of the SAT-Problem has lead to remarkable research results and powerful tools for SAT solving. In this chapter, we will focus in the DPLL algorithm, the basis of many SAT solvers.

2.1 The SAT-Problem

Given a formula in propositional logic, the question whether there exists a satisfying model is called the *boolean satisfiability problem*, or the SAT problem for short.

Definition 2.1 (SAT-Problem.) Let φ be a formula in propositional logic. The *boolean satisfiability problem* (SAT-Problem) asks whether there exists a satisfying model $\mathcal{M}$ for φ, i.e. $\mathcal{M} \vDash \varphi$.

The SAT problem is proven to be *NP-complete*. Given its NP-completeness, it is very unlikely that there exists any polynomial algorithm for SAT. Nevertheless, there exist algorithms that are able to solve many interesting SAT instances efficiently.

2.2 The DPLL Algorithm

In 1962 Martin **D**avis, Hilary **P**utnam, Donald W. **L**oveland and George **L**ogemann introduced the DPLL algorithm. Their approach still forms the basis for most modern SAT solvers.

The DPLL algorithm operates on formulas given in *conjunctive normal form* (CNF) and is a complete, backtracking-based *binary search* algorithm.

Definition 2.2 (Conjunctive Normal Form) A formula φ is in *conjunctive normal form* if it is a conjunction of clauses. A *clause* is a disjunction of literals.

Examples. The following formulas are in conjunctive normal form:

- $\varphi = (a \vee b \vee c) \wedge (a \vee \neg b)$,
- $\varphi = (a \vee \neg b) \wedge (\neg a \vee \neg b) \wedge (c \vee d) \wedge (a \vee \neg b \vee c)$, and
- $\varphi = a$.

We will introduce the basic version of the algorithm and add additional optimizations further on. In its basic version, the algorithm tries every possible assignment of truth values to the variables of the input formula. We can visualizes the search space as a full binary search tree:

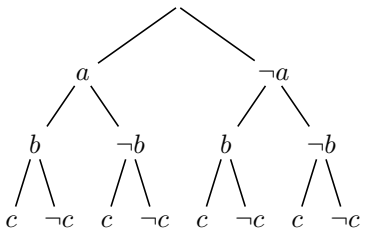


Figure 2.1: The search space for an input formula φ consisting of variables $a, b,$ and c.

The left-most leave of the tree in Figure 2.1 represents the model $\mathcal{M} = \{a = \top, b = \top, c = \top\}$. As large formulas will induce a large search space, optimization techniques focus on *trimming* the search space, i.e. not exploring paths that may not lead to a satisfying model. Additionally, most modern SAT solvers implement techniques that allow them to learn from wrong assignments. We will cover one such technique at the end of this chapter.

The Basic DPLL Algorithm – Backtracking Binary Search

The basic procedure for a recursive implementation of the DPLL algorithm is given in Listing 2.1. Note that the procedure in Listing 2.1 gives a rough overview of how the search space is traversed. Be aware that it does not cover all the details of the DPLL algorithm.

```
# sat(φ, {}) = True iff φ is satisfiable
# sat(φ, A) = True iff φ[A] is satisfiable
procedure sat(φ, A):
  if φ[A] = ⊥:
    return False
  if φ[A] = ⊤:    # φ is SAT, A is satisfying assignment
    return True

  # There are some unassigned variables left
  # Assign next variable
  l = pick unassigned variable
  if sat(φ, A ∪ {l = ⊤})
    return True
  if sat(φ, A ∪ {l = ⊥})
    return True
  return False
```

Listing 2.1: The basic recursive call of the DPLL algorithm

The algorithm starts the recursion with an empty assignment, $A := \{\}$, and calls `sat`(φ, A). In each recursive step, the procedure first checks whether φ evaluates to false under the current assignment A. If so, no extension of A will satisfy φ, and the recursive call returns `False`, effectively stopping traversing this path in the search space. Otherwise, if A is a satisfying assignment, we return `True`.

Since A is not a full assignment yet (the procedure would have otherwise already returned), the procedure picks one of the not yet assigned variables and continues recursively. In order to do so, the procedure first calls `sat`$(\varphi, A \cup \{l = \top\})$ and returns `True` if this extension is not falsifying, effectively continuing to traverse this branch. Otherwise, the procedure traverses the other branch of the search space and calls `sat`$(\varphi, A \cup \{l = \bot\})$. Again, the procedure returns `True`, if the new assignment is not falsifying. If both recursive calls return `False`, this part of the search space must not be traversed further. In case both recursive calls return `False` for the first call of the recursion, we report that φ is unsatisfiable. In order to keep track of this, the procedure keeps track of the so called *decision level*.

Definition 2.3 (Decision and Decision Level) Whenever the algorithm decides on a truth value for a new literal l it makes a *decision*. The number of decision currently assigned is called the *decision level*.

This basic version has seen many different extensions, all of which are aimed at reducing the time needed to determine the satisfiability of the input formula. Most of these extensions come in the form different *heuristics*. The first heuristics we are going to look at are so-called decision heuristics.

Decision Heuristic

SAT solvers that implement the DPLL algorithm use heuristics to decide which variable should be used to continue the search for a satisfying assignment. The order in which the variables are chosen heavily influences the time needed to decide whether the input formula is satisfiable. The efficiency of these heuristics is measured empirically, by executing the algorithm on large sets of benchmarks.

We will discuss one simple heuristic, as well as the approach we are going to use in this course. You can read more about different heuristics in Chapter 2.2.5 of [1].

Dynamic Largest Individual Sum. In each iteration, count for each potential assignment of truth value to a not yet assigned variable the number of clauses it appears in. Then, pick the variable and truth value, such that the most clauses become satisfied. Although this approach might seem like a good heuristic, it introduces a large overhead as it is proportional to the number of clauses that are not yet satisfied.

Predefined Order. *In this course*, we will always define the order for decision making for every example. E.g. *lexicographical order, positive phase first*: Choose the variables in lexicographical order and assign the truth value "true" first. We denote this as follows: $a < \neg a < b < \neg b < c < \neg c < \dots$.

Notation

For the remainder of this chapter, we will use shorthand notations for both the input formula φ in CNF and the current assignment.

A clause can be represented as a set containing its literals, e.g. $(\neg a \vee b) \equiv \{\neg a, b\}$, since the logical connectives are clear by definition. The input φ can then be expressed as a set of sets. For instance, we express the input

$$\varphi := (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee \neg a).$$

via

$$C_\varphi := \{\{\neg a, b\}, \{\neg b, c\}, \{\neg c, \neg a\}\}.$$

Similarly, we will use a shorthand in the set notation of assignments. If a variable is assigned the truth value "true", we represent it as the variable itself, i.e. $\{a = \top\} \equiv \{a\}$. Similarly, we represent with $\{\neg a\}$ the assignment $\{a = \bot\}$.

Definition 2.4 (State of a clause under A.) A clause is *satisfied* if at least one of its literals are satisfied under A, *conflicting* if all of its literals are not satisfied under A, and *unresolved* otherwise.

These notations allows us to compactly represent φ and quickly compute φ under different assignments A when executing the DPLL algorithm. In order to compute $\varphi[A \cup \{l\}]$ using set representation we perform two steps:

1. Remove all clauses that contain l, since they are satisfied.
2. Remove all literals $\neg l$ from all remaining clauses, because these literals evaluate to *false*. Report any empty clause as conflicting.

The computation of $\varphi[A \cup \{\neg l\}]$ follows analogously.

Checking for falsifying assignments. The input φ under an assignment A evaluates to *false*, if using set representation at least one *clause is conflicting.*

Checking for satisfying assignments. Finally, we can efficiently check whether φ under assignment A evaluates to *true*. It does so if the resulting *set of clauses becomes empty*, i.e., all clauses evaluate to *true* and are removed from C_φ.

Exercise 2.1

Given $\varphi := (a \vee b \vee c) \wedge d$ and $A := \{a\}$. Compute $\varphi[A]$ using set representation. State any satisfied or conflicting clauses.

Solution. Expressing φ in set representation gives us

$$C_\varphi := \{\{a, b, c\}, \{d\}\}.$$

Since $a \in \{a, b, c\}$, we remove the first clause from C_φ, resulting in $C_\varphi := \{\{d\}\}$ or $\varphi[A] = d$, and we report the first clause to be satisfied.

Exercise 2.2

Given $\varphi := (\neg a \vee b \vee c) \wedge d$ and $A := \{a, \neg d\}$. Compute $\varphi[A]$ using set representation. State any satisfied or conflicting clauses.

Solution. Expressing φ in set representation gives us

$$C_\varphi := \{\{\neg a, b, c\}, \{d\}\}.$$

Under A we remove $\neg a$ from the first clause and d from the second clause, resulting in $C_\varphi := \{\{b, c\}, \{\}\}$ or $\varphi[A] := (b \vee c) \wedge \bot$. We can therefore report the first clause to be unresolved and the second to be conflicting.

Tabular Execution of the DPLL Algorithm

In order to study the DPLL algorithm in more detail, we will use a tabular representation to complete exercises in this course. The tabular representation in Table 2.1 shows the different components needed.

Each iteration of the algorithm is represented by a single column in the table. The different rows hold specific information: the current decision level, the literals currently assigned, the set representation of the clauses under the current assignment and the next decision chosen by the algorithm.

To execute the algorithm, we follow these steps:

Step i) Enter the input clauses in set notation into the first column of the table.

Step ii) Start with the empty assignment $A = \{\}$. The algorithm now starts with the recursive procedure.

Step iii) Evaluate φ under the current assignment A:

- Clauses that are satisfied under A are replaced by a ✓,
- unresolved clauses are updated and entered into their respective cell, and
- conflicting clauses are replaced by a $\{\}$ ✗. We call this a *conflict*.

Step	1	2	3	4	5	6	7
Decision Level							
Assignment							
Cl. 1:							
Cl. 2:							
...							
Cl. n:							
Decision							

Table 2.1: The table used for DPLL exercises.

Step iv) If all clauses are satisfied, we report `SAT` and A as a satisfying model.

Step v) If no clause is conflicting, we continue with step Step vii).

Step vi) If there is at least one conflicting clause under the current assignment A, we backtrack. When backtracking the algorithm removes the last decisions and the decision level is reduced by the number of reverted decisions.

- If the conflict occured in decision level 0, the algorithm stops and reports `UNSAT`.
- Otherwise, the algorithm continues with step Step vii) and chooses the next combination of literal and truth value.

Step vii) Update the current assignment A with the next literal l. Increment the decision level by 1.

Exercise 2.3

Use the DPLL Algorithm to determine whether or not the formula

$$\varphi := (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee \neg a)$$

is satisfiable.
Use a lexicographical order starting with the positive value as the decision heuristic: $a < \neg a < b < \neg b < c < \neg c < \dots$

Solution. We start by representing φ in the set representation,

$$C_\varphi := \{\{\neg a, b\}, \{(\neg b, c\}, \{\neg c, \neg a\},$$

and enter the individual clauses into the table.

In the following, we will discuss more heuristics that improve the run time of the DPLL algorithm.

Step	1	2	3	4	5	6	7	8	9
Decision Level	0	1	2	3	3	2	1	2	3
Assignment	-	a	a, b	a, b, c	$a, b, \neg c$	$a, \neg b$	$\neg a$	$\neg a, b$	$\neg a, b, c$
Cl. 1: $\neg a, b$	$\neg a, b$	b	✓	✓	✓	{} ✗	✓	✓	✓
Cl. 2: $\neg b, c$	$\neg b, c$	$\neg b, c$	c	✓	{} ✗	✓	$\neg b, c$	c	✓
Cl. 3: $\neg c, \neg a$	$\neg c, \neg a$	$\neg c$	$\neg c$	{} ✗	✓	$\neg c$	✓	✓	✓
Decision	a	b	c	$\neg c$	$\neg b$	$\neg a$	b	c	SAT

In step 0 of the algorithm, we pick a as our first assignment. In step 1 this choice is added to A and we evaluate the clauses accordingly.

In step 4, after assigning c, the third clause becomes conflicting and we reach our first conflict. We backtrack and flip the truth value assignment for and assign $\neg c$. Once again, this leads to a conflicting clause in step 5 and we revert the last two decisions, namely $\neg c$ and b. The algorithm then picks $\neg b$.
We continue until step 9, in which all clauses are satisfied and we report $A = \{\neg a, b, c\}$ as a satisfying model.
Figure 2.2 illustrates the binary search we have performed during the search and annotates the decision levels.

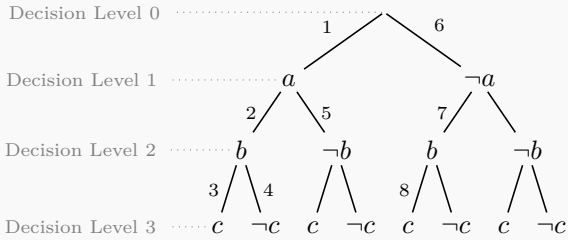


Figure 2.2: Binary Search Tree

2.2.1 DPLL with Boolean Constraint Propagation

As a first optimization we will introduce *Boolean Constraint Propagation* (BCP). To discuss this heuristic we first need to extend Definition **??** with *unit clauses*:

Definition 2.5 (Unit Clause.) A clause is said to be a *unit clause*, or *unary*, under some assignment A if the clause is not satisfied by A and all but one of its variables are assigned in A.

The key observation is that in order to extend A to a satisfying assignment for any formula that contains a unit clause, we must assign the according literal:

For any unit clause $c = \{l\}$ the next assignment must include l, resulting in

$$A = A \cup \{l\}.$$

In order to add this heuristic to our approach, we modify step Step vii) by executing BCP before making a next decision:

Step vii) Chose a literal as update for the next assignment:

- If there is a unit clause, execute BCP and extend A with the literal of the unit clause.
- Otherwise, update the current assignment A with the next literal l. Increment the decision level by 1.

Note that, when applying BCP, *the decision level is not increased.* We say that an assignment of a variable that is made due to a unit clause is an implication and therefore *not a decision.*

Exercise 2.4

Use the DPLL Algorithm with BCP to determine whether or not the formula

$$\varphi := (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee \neg a)$$

is satisfiable. Use a lexicographical order starting with the positive value as the decision heuristic: $a < \neg a < b < \neg b < c < \neg c < \ldots$

Solution. As above, we start by representing φ in set notaton and list the clauses in the first column.

Step	1	2	3	4	5	6	7
Decision Level	0	1	1	1	1	2	2
Assignment	-	a	a, b	a, b, c	$\neg a$	$\neg a, b$	$\neg a, b, c$
Cl. 1: $\neg a, b$	$\neg a, b$	b	✓	✓	✓	✓	✓
Cl. 2: $\neg b, c$	$\neg b, c$	$\neg b, c$	c	✓	$\neg b, c$	c	✓
Cl. 3: $\neg c, \neg a$	3	$\neg c$	$\neg c$	$\{\}$ ✗	✓	✓	✓
BCP	-	b	c	-	-	c	-
Decision	a	-	-	$\neg a$	b	-	SAT

In this example we apply BCP in step 2, 3, and 6. The decision level is not incremented when applying BCP. As we can see, the algorithm is able to find a satisfying assignment with a fewer number of steps.

2.2.2 DPLL with Pure Literals

The next optimization is the pure literal rule and is one of the standard techniques used in DPLL-based SAT solvers.

Definition 2.6 (Pure Literal.) A literal is *pure* if its negation does not appear in the formula.

Example. Consider the set of clauses $C_\varphi := \{\{a, \neg b, c\}\{a, \neg c\}, \{b, \neg c\}\}$. The literal a is pure in φ, since $\neg a$ does not appear in any clause.

Similarly to BCP, we can deduce that assigning the negation of a pure literal cannot help in the search for a satisfying assignment. On the other hand, by assigning a pure literal all clauses containing it will immediately be satisfied.

We can therefore further extend the DPLL algorithm, by modifying step Step vii) once more:

Step vii) Chose a literal as update for the next assignment:

- If there is a unit clause, execute BCP and extend A with the literal of the unit clause.
- If there is a pure literal, execute PL and extend A with the pure literal.
- Otherwise, update the current assignment A with the next literal l. Increment the decision level by 1.

As for BCP, *the decision level is not increased* when applying the pure literal rule.

Exercise 2.5

Use the DPLL Algorithm with BCP and PL to determine whether or not the formula

$$\varphi := (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee \neg a)$$

is satisfiable. Use a lexicographical order starting with the positive value as the decision heuristic: $a < \neg a < b < \neg b < c < \neg c < \ldots$

Solution.

Step	1	2	3
Decision Level	0	0	0
Assignment	-	$\neg a$	$\neg a, \neg b$
Cl. 1: $\neg a, b$	$\neg a, b$	✓	✓
Cl. 2: $\neg b, c$	$\neg b, c$	$\neg b, c$	✓
Cl. 3: $\neg c, \neg a$	$\neg c, \neg a$	✓	✓
BCP	-	-	-
PL	$\neg a$	$\neg b$	-
Decision	-	-	SAT

The formula φ contains the pure literal $\neg a$, therefore the pure-literal rule is applied in the first step and satisfies the clauses 1 and 3. The single remaining second clause gives us two pure literals: $\neg b$ and c. The algorithm picks $\neg b$. The assignment $A = \{\neg a, \neg b\}$ satisfies all clauses.

2.2.3 Conflict-driven Clause Learning

We now come to the final optimization technique, *conflict-driven clause learning* (CDCL). The principal idea of CDCL is to learn from conflicts that the algorithm encountered so far. As the name suggests, the aim is to *learn new clauses* and add them to the set of clauses. The newly added clauses influence the choice of the next assignment and keeps the algorithm from encoutering the same conflict. There are different methods to learn new clauses. In this course we will focus on a simple approach.

SAT solvers keep track of decisions that they have made and implications that occured due to BCP. This information is tracked via a *conflict graph*, or *implication graph*.

Definition 2.7 (Conflict Graph) A *conflict graph* is a directed, acyclic graph $G(V, E)$, where:

- $V = V_{dec} \cup V_{imp} \cup \{\bot\}$ is a set of nodes representing the partial assignment. V_{dec} is a set of root nodes that represent all decisions. V_{imp} consists of implications that occured due to BCP. The special node $\bot$ represents the conflict.
- E is a set of edges. Labeled edges connect two nodes from $V_{dec} \cup V_{imp}$ and denote the clause as the reason for the implication. Unlabeled edges connect nodes from $V_{dec} \cup V_{imp}$ to $\bot$, denoting the clauses that lead to the conflict.

A conflict graph is constructed whenever the algorithm encounters a conflict, following these steps:

1. For every decision, create a root node that is labeled with that decision.
2. For every implication due to BCP, create a new node that is labeled the implication. Connect the assignments that caused the clause to become unary with the newly added implication. Label all edges with the index of the unit clause. If the set of clauses contains a unary clause, i.e. no decision is needed for this implication, add the according node with a dangling edge.
3. Connect all implications corresponding to the conflict with $\bot$.

Exercise 2.6

Execute the DPLL algorithm with BCP and PL for the set of clauses C_φ until the first conflict occurs. Draw the conflict graph.

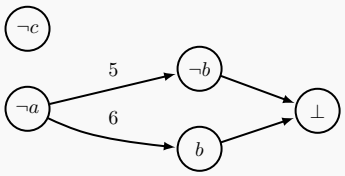
$$C_\varphi := \{\{a, \neg c\}, \{b, \neg c\}, \{\neg a, \neg b, c\}, \{\neg a, \neg b\}, \{\neg a, b\}\{a, \neg b\}, \{a, b\}.$$

Use decision heuristic $\neg c < c < \neg a < a < \neg b < b < \ldots$.

Solution.

Step	1	2	3	4
Decision Level	0	1	2	2
Assignment	-	$\neg c$	$\neg c, \neg a$	$\neg c, \neg a, \neg b$
Cl. 1: $a, \neg c$	$a, \neg c$	✓	✓	✓
Cl. 2: $b, \neg c$	$b, \neg c$	✓	✓	✓
Cl. 3: $\neg a, \neg b, c$	$\neg a, \neg b, c$	$\neg a, \neg b$	✓	✓
Cl. 4: $\neg a, \neg b$	$\neg a, \neg b$	$\neg a, \neg b$	✓	✓
Cl. 5: $a, \neg b$	$a, \neg b$	$a, \neg b$	$\neg b$	✓
Cl. 6: a, b	a, b	a, b	b	$\{\}$ ✗
BCP	-	-	$\neg b$	-
PL	-	-	-	-
Decision	$\neg c$	$\neg a$	-	-

We encounter the first conflicting clause in step 4.
In order to draw the conflict graph, we start by adding two nodes for the decisions of the current assignment: $\neg c$ and $\neg a$. The decision $\neg a$ caused clauses 5 and 6 to become unary, we therefore add $\neg b$ and b as implications and connect the nodes. The edges are labeled according to the unit clauses. Since these two clauses are the conflicting clauses, we add the conflict node and connect the nodes representing $\neg b$ and b.



Clause Learning via Resolution Proofs

CDCL uses the generated conflict graphs to derive new clauses. As mentioned above, SAT solvers implement different strategies for the computation of learned clauses from the conflict graph. *In this course*, we apply an approach based on resolution proofs.

Definition 2.8 (Resolution Rule.) Let $c_1 = (\varphi \vee l)$ and $c_2 = (\psi \vee \neg l)$ be two

clauses, where φ and ψ denote disjunctions of arbitrary literals. Then the clause $\varphi \vee \psi$ is implied by $c_1 \wedge c_2$.

The resolution rule is a derived natural deduction rule and can be written as follows:

$$\frac{(\varphi \vee l) \qquad (\neg l \vee \psi)}{(\varphi \vee \psi)}$$

Definition 2.9 (Resolution Proof.) A *resolution proof* derives a new clause from a set of clauses by applying the resolution rule.

The resolution proof for a new learned clause can easily be generated by traversing the conflict graph from the *conflict node* to the *root nodes*. The clauses that are represented by the labels in the graph are iteratively used to apply the resolution rule.

Example. We want to state a resolution proof to compute a new clause for Exercise 2.6. We iteratively apply the resolution rule to the clauses in the conflict graph. Since the conflict graph contains only clauses 5 and 6 we apply the rule and learn the new clause $\{a\}$.

$$\frac{5.\ a \vee \neg b \qquad 6.\ a \vee b}{a}$$

Figure 2.3: The resolution proof for conflict graph in Exercise 2.6

Backtracking Level: After a conflict is reached and a new learned clause is added, the DPLL algorithm needs to perform backtracking. Again, different strategies exists on which level the SAT solver backtracks after adding a learned clause.

In this course we apply the following rule for deciding the backtracking level: After reaching a conflict and adding a learned clause, we backtrack to the level where the *newly added clause becomes a unit clause*. This way, the newly added clause can immediately be used with BCP.

Exercise 2.7

Use the DPLL algorithm with BCP, PL and CDCL to determine whether the set of clauses C_φ is satisfiable. Draw the conflict graph and state the resolution proof for any conflict.

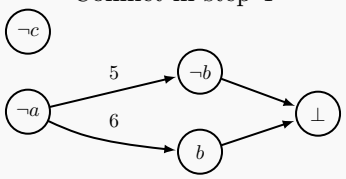
$$C_\varphi := \{\{a, \neg c\}, \{b, \neg c\}, \{\neg a, \neg b, c\}, \{\neg a, \neg b\}, \{\neg a, b\}\{a, \neg b\}, \{a, b\}.$$

Use decision heuristic $\neg c < c < \neg a < a < \neg b < b < \ldots$.

Solution.

Step	1	2	3	4
Decision Level	0	1	2	2
Assignment	-	$\neg c$	$\neg c, \neg a$	$\neg c, \neg a, \neg b$
Cl. 1: $a, \neg c$	$a, \neg c$	✓	✓	✓
Cl. 2: $b, \neg c$	$b, \neg c$	✓	✓	✓
Cl. 3: $\neg a, \neg b, c$	$\neg a, \neg b, c$	$\neg a, \neg b$	✓	✓
Cl. 4: $\neg a, \neg b$	$\neg a, \neg b$	$\neg a, \neg b$	✓	✓
Cl. 5: $a, \neg b$	$a, \neg b$	$a, \neg b$	$\neg b$	✓
Cl. 6: a, b	a, b	a, b	b	{} ✗
BCP	-	-	$\neg b$	-
PL	-	-	-	-
Decision	$\neg c$	$\neg a$	-	-

Conflict in step 4



$$\frac{5.\ a \vee \neg b \qquad 6.\ a \vee b}{a}$$

Step	5	6	7
Decision Level	1	1	1
Assignment	$\neg c$	$\neg c, a$	$\neg c, a, \neg b$
Cl. 1: $a, \neg c$	✓	✓	✓
Cl. 2: $b, \neg c$	✓	✓	✓
Cl. 3: $\neg a, \neg b, c$	$\neg a, \neg b$	$\neg b$	✓
Cl. 4: $\neg a, \neg b$	$\neg a, \neg b$	$\neg b$	✓
Cl. 5: $a, \neg b$	$a, \neg b$	✓	✓
Cl. 6: a, b	a, b	✓	✓
Cl. 7: a	a	✓	✓
BCP	a	$\neg b$	-
PL	-	-	-
Decision	-	-	SAT

Proofs for Unsatisfiability

Finally, in the case the input formula φ is unsatisfiable, we want to give a concise proof that shows its unsatisfiability. When a conflict occurs at decision level 0 we state a resolution proof in the same manner as when learning a new clause. Note that this can also be viewed as learning the *empty clause*, which cannot be satisfied and the whole formula is therefore unsatisfiable.

Example. Consider the second conflict in Exercise 2.8. The proof in Figure 2.4 shows the same proof. We start by applying the resolution rule using clauses 2 and 4, according to the conflict graph. We deduce that the algorithm needs to set $\neg a$ for any satisfying assignment. In a previous conflict we have learned that the

algorithm also needs to assign a and we deduce $\bot$, resulting in the proof that the input is unsatisfiable.

$$\dfrac{\dfrac{2.\ \neg a \vee b \qquad 4.\ \neg a \vee \neg b}{\neg a} \qquad 5.\ a}{\bot}$$

Figure 2.4: The resolution proof for step 6 in 2.8 showing that the formula unsatisfiable.

The DPLL Algorithm with CDCL

We conclude this chapter with the final algorithm. In order to incorporate conflict-driven clause learning we need to adapt step Step vi). Instead of backtracking and choosing the next assignment according to the decision heuristic, we learn a new clause.

Step i) Enter the input clauses in set notation into the first column of the table.

Step ii) Start with the empty assignment $A = \{\}$. The algorithm now starts with the recursive procedure.

Step iii) Evaluate φ under the current assignment A:

- Clauses that are satisfied under A are replaced by a ✓,
- unresolved clauses are updated and entered into their respective cell, and
- conflicting clauses are replaced by a $\{\}$ ✗. We call this a *conflict*.

Step iv) If all clauses are satisfied, we report `SAT` and A as a satisfying model.

Step v) If no clause is conflicting, we continue with step Step vii).

Step vi) If there is at least one conflicting clause under the current assignment A we process the conflict. We start by drawing the corresponding conflict graph and formulate a resolution proof.

- If we have learned a new clause, the algorithm backtracks and reverts decisions, such that the newly learned claused triggers BCP in the next step.
- If the conflict occured at decision level 0, we have proven that the input is unsatisfiable and we stop.

Step vii) Chose a literal as update for the next assignment:

- If there is a unit clause, execute BCP and extend A with the literal of the unit clause.
- If there is a pure literal, execute PL and extend A with the pure literal.
- Otherwise, update the current assignment A with the next literal l. Increment the decision level by 1.

Exercise 2.8

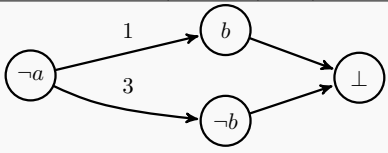
Use the DPLL algorithm with BCP, PL and CDCL to determine whether the set of clauses C_φ is satisfiable. Draw the conflict graph and state the resolution proof for any conflict.

$$C_\varphi := \{\{a, b\}, \{a, \neg b\}, \{\neg a, b\}, \{\neg a, \neg b\}$$

Use decision heuristic $\neg a < a < \neg b < b < \ldots$.

Solution.

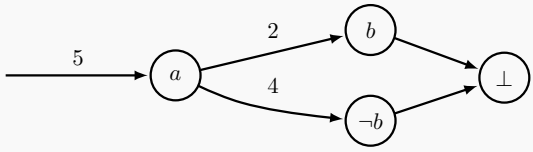
Step	1	2	3	4	5	6
Decision Level	0	1	1	0	0	0
Assignment	-	$\neg a$	$\neg a, \neg b$	-	a	$a, \neg b$
Cl. 1: a, b	a, b	b	$\{\}$ ✗	a, b	✓	✓
Cl. 2: $\neg a, b$	$\neg a, b$	✓	✓	$\neg a, b$	b	$\{\}$ ✗
Cl. 3: $a, \neg b$	$a, \neg b$	$\neg b$	✓	$a, \neg b$	✓	✓
Cl. 4: $\neg a, \neg b$	$\neg a, \neg b$	✓	✓	$\neg a, \neg b$	$\neg b$	✓
Cl. 5: a (LC)	-	-	-	a	✓	✓
BCP	-	$\neg b$	-	a	$\neg b$	-
PL	-	-	-	-	-	-
Decision	$\neg a$	-	-	-	-	UNSAT



$$\frac{1.\ a \vee b \qquad 3.\ a \vee \neg b}{a}$$

After the first conflict we can deduce that we need to assign a and add it as a learned clause. We backtrack, such that we can use BCP and therefore move back to decision level 0.

In the next step the unary clause 4 triggers BCP again and we reach another conflict.



$$\frac{\dfrac{2.\ \neg a \vee b \qquad 4.\ \neg a \vee \neg b}{\neg a} \qquad 5.\ a}{\bot}$$

Exercise 2.9

Use the DPLL Algorithm with BCP, PL and CDCL to determine whether or not the formula

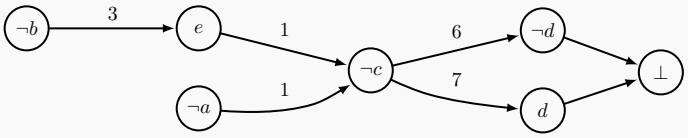
$$\varphi := (a \vee \neg c \vee \neg e) \wedge (\neg a \vee \neg e) \wedge (b \vee e) \wedge (\neg b \vee d \vee e) \wedge (\neg b \vee \neg d) \wedge (c \vee \neg d) \wedge (c \vee d)$$

is satisfiable. Use a lexicographical order starting with the negative value as the decision heuristic: $\neg a < a < \neg b < b < \neg c < c < \dots$

Solution.

Step	1	2	3	4	5	6
Decision Level	0	1	2	2	2	2
Assignment	-	$\neg a$	$\neg a, \neg b$	$\neg a, \neg b, e$	$\neg a, \neg b, e, \neg c$	$\neg a, \neg b, e, \neg c, \neg d$
Cl. 1: $a, \neg c, \neg e$	$a, \neg c, \neg e$	$\neg c, \neg e$	$\neg c, \neg e$	$\neg c$	✓	✓
Cl. 2: $\neg a, \neg e$	$\neg a, \neg e$	✓	✓	✓	✓	✓
Cl. 3: b, e	b, e	b, e	e	✓	✓	✓
Cl. 4: $\neg b, d, e$	$\neg b, d, e$	$\neg b, d, e$	✓	✓	✓	✓
Cl. 5: $\neg b, \neg d$	$\neg b, \neg d$	$\neg b, \neg d$	✓	✓	✓	✓
Cl. 6: $c, \neg d$	$c, \neg d$	$c, \neg d$	$c, \neg d$	$c, \neg d$	$\neg d$	✓
Cl. 7: c, d	c, d	c, d	c, d	c, d	d	{} ✗
BCP	-	-	e	$\neg c$	$\neg d$	-
PL	-	-	-	-	-	-
Decision	$\neg a$	$\neg b$	-	-	-	-

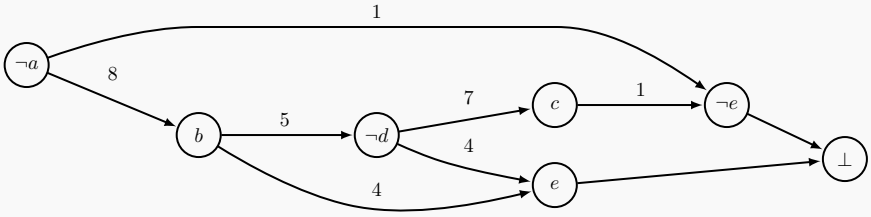
Conflict in step 6



$$\dfrac{\dfrac{\dfrac{6.\ c \vee \neg d \qquad 7.\ c \vee d}{c} \qquad 1.\ a \vee \neg c \vee \neg e}{a \vee \neg e} \qquad 3.\ b \vee e}{a \vee b}$$

Step	7	8	9	10	11
Decision Level	1	1	1	1	1
Assignment	$\neg a$	$\neg a, b$	$\neg a, b, \neg d$	$\neg a, b, \neg d, c$	$\neg a, b, \neg d, c, \neg e$
Cl. 1: $a, \neg c, \neg e$	$\neg c, \neg e$	$\neg c, \neg e$	$\neg c, \neg e$	$\neg e$	✓
Cl. 2: $\neg a, \neg e$	✓	✓	✓	✓	✓
Cl. 3: b, e	b, e	✓	✓	✓	✓
Cl. 4: $\neg b, d, e$	$\neg b, d, e$	d, e	e	e	$\{\}$ ✗
Cl. 5: $\neg b, \neg d$	$\neg b, \neg d$	$\neg d$	✓	✓	✓
Cl. 6: $c, \neg d$	$c, \neg d$	$c, \neg d$	✓	✓	✓
Cl. 7: c, d	c, d	c, d	c	✓	✓
Cl. 8: a, b	b	✓	✓	✓	✓
BCP	b	$\neg d$	c	$\neg e$	-
PL	-	-	-	-	-
Decision	-	-	-	-	-

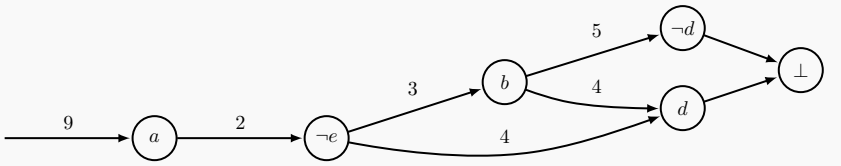
Conflict in step 11



$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{1.\ a \vee \neg c \vee \neg e \qquad 4.\ \neg b \vee d \vee e}{a \vee \neg c \vee \neg b \vee d} \qquad 7.\ c \vee d}{a \vee \neg b \vee d} \qquad 5.\ \neg b \vee \neg d}{a \vee \neg b} \qquad 8.\ a \vee b}{a}$$

Step	12	13	14	15	16
Decision Level	0	0	0	0	0
Assignment	-	a	$a, \neg e$	$a, \neg e, b$	$a, \neg e, b, \neg d$
Cl. 1: $a, \neg c, \neg e$	$a, \neg c, \neg e$	✓	✓	✓	✓
Cl. 2: $\neg a, \neg e$	$\neg a, \neg e$	$\neg e$	✓	✓	✓
Cl. 3: b, e	b, e	b, e	b	✓	✓
Cl. 4: $\neg b, d, e$	$\neg b, d, e$	$\neg b, d, e$	$\neg b, d$	d	$\{\}$ ✗
Cl. 5: $\neg b, \neg d$	$\neg b, \neg d$	$\neg b, \neg d$	$\neg b, \neg d$	$\neg d$	✓
Cl. 6: $c, \neg d$	$c, \neg d$	$c, \neg d$	$c, \neg d$	$c, \neg d$	✓
Cl. 7: c, d	c, d	c, d	c, d	c, d	c
Cl. 8: a, b	a, b	✓	✓	✓	✓
Cl. 9: a	a	✓	✓	✓	✓
BCP	a	$\neg e$	b	$\neg d$	-
PL	-	-	-	-	-
Decision	-	-	-	-	UNSAT

Conflict in step 16



$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{4.\ \neg b \vee d \vee e \qquad 5.\ \neg b \vee \neg d}{\neg b \vee e} \qquad 3.\ b \vee e}{e} \qquad 2.\ \neg a \vee \neg e}{\neg a} \qquad 9.\ a}{\bot}$$

This chapter is based on:
D. Kroening and O. Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition.* Springer, 2016.

List of Definitions

Bibliography

[1] D. Kroening and O. Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition.* Springer, 2016.