Logic for knowledge representation, learning, and inference

Luciano Serafini

Contents

Chapter 1. Decision Procedures in Propositional Logic 5
1. Model checking 5
2. Satisfiability checking 7
3. Normal forms 8
4. Davis-Putnam-Logemann-Loveland Procedure (DPLL) 15
5. Sat Solvers 19
6. Exercises 20

Bibliography 31

CHAPTER 1

Decision Procedures in Propositional Logic

In the previous chapter we have presented four problems that involves propositional formulas and interpretations. In this chapter we analize them in details. We will see that the problems of checking the validity of a formula, and the problem of cheking that a formula is a logical consequence of a set of formulas can be rewritten in a problem of checking satisfiability of a set of formulas. Therefore, after a first part in which we will consider the problem of model checking, we will dedicate most of the chapter to the problem of satisfiability.

1. Model checking

The first and simplest problem that one has to solve in propositional logic is to compute the truth value of a formula with respecrt to a given interpretation $\mathcal{I}$.

Definition 1.1 (Model checking Problem). *Given a formula ϕ on a set of primitive propositions $\mathcal{P}$, and an interpretation (truth assignment) $\mathcal{I}$ of $\mathcal{P}$, the* model checking problem *is the problem of checking if*

$$\mathcal{I} \models \phi$$

An alternative formulation of the model cheking problem, is the problem of computing the truth value of a formula ϕ w.r.t., an interpretation $\mathcal{I}$ of the propositional variables of ϕ.

1.1. Decision procedure. A model checking decision procedure, MCDP is an algorithm that checks if a formula ϕ is satisfied by an interpretation $\mathcal{I}$. Namely

$$\begin{aligned} \mathrm{MCDP}(\phi, \mathcal{I}) = \mathit{true} & \quad \text{if and only if} \quad \mathcal{I} \models \phi \\ \mathrm{MCDP}(\phi, \mathcal{I}) = \mathit{false} & \quad \text{if and only if} \quad \mathcal{I} \not\models \phi \end{aligned}$$

The procedure of model checking returns for all inputs either *true* or *false* since for all models $\mathcal{I}$ and for all formulas ϕ, we have that either $\mathcal{I} \models \phi$ or $\mathcal{I} \not\models \phi$.

A simple way to check if $\mathcal{I} \models \phi$ is the following:

(1) Replace each occurrence of a propositional variables in ϕ with the truth value assigned by $\mathcal{I}$. I.e. replace each p with $\top$ is $\mathcal{I}(p) = \text{True}$ and with $\bot$ is $\mathcal{I}(p) = \text{False}$;
(2) Recursively apply the following rewriting rules:

$$\begin{array}{lllll} \neg\top \Rightarrow \bot & \top \wedge \top \Rightarrow \top & \top \vee \top \Rightarrow \top & \top \rightarrow \top \Rightarrow \top & \top \equiv \top \Rightarrow \top \\ \neg\bot \Rightarrow \top & \top \wedge \bot \Rightarrow \bot & \top \vee \bot \Rightarrow \top & \top \rightarrow \bot \Rightarrow \bot & \top \equiv \bot \Rightarrow \bot \\ & \bot \wedge \top \Rightarrow \bot & \bot \vee \top \Rightarrow \top & \bot \rightarrow \top \Rightarrow \top & \bot \equiv \top \Rightarrow \bot \\ & \bot \wedge \bot \Rightarrow \bot & \bot \vee \bot \Rightarrow \bot & \bot \rightarrow \bot \Rightarrow \top & \bot \equiv \bot \Rightarrow \top \end{array}$$

The complexity of this procedure is linear in the number of the connectives that appear in the formula ϕ. More specifically, the algorithm terminates in n steps where n is the number of connectives that appear in ϕ.

EXAMPLE 1.1. *Let us consider the formula* ϕ

$$\phi = p \vee (q \to r)$$

and the interpretation $\mathcal{I}$ *with*

$$\mathcal{I}(p) = \mathit{False} \qquad \mathcal{I}(q) = \mathit{False} \qquad \mathcal{I}(r) = \mathit{True}$$

To check if $\mathcal{I} \models \phi$ *replace,* p*,* q*, and* r *in* ϕ *with* $\mathcal{I}(p)$*,* $\mathcal{I}(q)$ *and* $\mathcal{I}(r)$*, obtaining*

$$\bot \vee (\bot \to \top)$$

and then recursively apply the reduction rules

$$\begin{array}{l} \bot \vee (\bot \to \top) \\ \bot \vee \top \\ \top \end{array}$$

Since we obtain $\top$ *we can conclude that* $\mathcal{I} \models \phi$*.*

This method is not the most efficient one. One can notice that in many rewriting rules if one for the binary connectives $\wedge$,$\vee$, and $\to$, if one knows the value of one of the arguments it is not necessary to evaluate the other. For instance in evaluating $\phi \vee \psi$ if we already know that ϕ is true, we don't need to evaluate ψ, since we already know that the disjunction is true. Similarly in $\phi \wedge \psi$, if we have evaluated ϕ to be false, then we do not need to evaluate ψ since we already know, that independently from the evaluation of ψ the conjunction will be false. Notice that, this simplification is not possible for the $\equiv$ connective, since to evaluate the truth value of $\phi \equiv \psi$ knowing the truth value of ϕ is not sufficient to predict the truth value of the entire formula; we indeed have to evaluate ψ and check that the two truth values coincides.

The rewriting rules for $\wedge$, $\vee$ and $\to$ therefore can be rewritten as follows:

$$\begin{array}{ccc} \top \wedge \top \Rightarrow \top & \top \vee * \Rightarrow \top & * \to \top \Rightarrow \top \\ * \wedge \bot \Rightarrow \bot & * \vee \top \Rightarrow \top & \top \to \bot \Rightarrow \bot \\ \bot \wedge * \Rightarrow \bot & \bot \vee \bot \Rightarrow \bot & \bot \to * \Rightarrow \top \end{array}$$

where "$*$" denotes either $\top$ or $\bot$ and therefore the corresponding expression does not need to be evaluated.

1.2. Formulas as boolean functions. A propositional formula ϕ on the set of propositional variables $\mathcal{P}$ specifies a boolean function

$$f_\phi : \{0,1\}^{|\mathcal{P}|} \to \{0,1\}$$

f_ϕ is a function that takes a vector of n 0-1 values (corresponding to a truth assignment) where $n = |\mathcal{P}|$ and returns either 1 or 0 depending of the fact that the corresponding truth assignment satisfies or does not satisfies the formula. More formually, if $\mathcal{P} = \{p_1, \ldots, p_n\}$, for every vector $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ we define

$\mathcal{I}_{\boldsymbol{x}}$ as the interpretation of $\mathcal{P}$ that assigns $\mathcal{I}_{\boldsymbol{x}}(p_i) = x_i$. With this convention the function f_ϕ can be defined

$$f_\phi(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \mathcal{I}_{\boldsymbol{x}} \models \phi \\ 0 & \text{if } \mathcal{I}_{\boldsymbol{x}} \not\models \phi \end{cases}$$

Notice that, if ϕ is equivalent to ψ then f_ϕ and f_ψ are the same functions.

PROPOSITION 1.1. *If $|\mathcal{P}| = n$ then there are at most 2^{2^n} boolean functions that can be defined in terms of a formulas with propositinal variables in $\mathcal{P}$.*

PROOF. The number of functions from a finite set M to another finite set N are n^m, where m and n are the number of elements (the cardinality) of the sets M and N respectively. Indeed for every element $x \in M$ whe have n different options to define $f(x)$ since $f(x) \in N$. So we have n options for the first element of M, other n options for the second elements, and so on. In total we have n^m alternatives. Therefore the number of distinct functions form $\{0,1\}^n$ to $\{0,1\}$ are equal to [1]

$$|\{0,1\}|^{|\{0,1\}^n|} = 2^{2^n}$$

Notice that $\{0,1\}$ contains 2 elements, and $\{0,1\}^n$ contains 2^n elements. □

The second question one should answer is the following: given an arbitrary function $f : \{0,1\}^n \to \{0,1\}$ is there a formula on the set $\mathcal{P}$ of n propositional fariables such that f_ϕ is equal to f?. The answer is yes, and it is stated in the following proposition.

PROPOSITION 1.2. *For every function $f : \{0,1\}^n \to \{0,1\}$, there is a formula ϕ with at most n propositional variables, such that f_ϕ is equal to f.*

PROOF. Let $p_1, \dots, p_n$ be n distinct propositional variables. We define the formula ϕ as follows: For every $\boldsymbol{x} \in \{0,1\}^n$ let us define the formula $\phi_{\boldsymbol{x}}$ as the conjunction of p_i if $x_i = 1$ and $\neg p_i$ if $x_i = 0$. We can then define the formula

$$\phi = \bigvee_{f(\boldsymbol{x})=1} \phi_{\boldsymbol{x}}$$

Notice that the formula ϕ is satisfied by the assignments $\mathcal{I}_{\boldsymbol{x}}$ such that $f(\boldsymbol{x}) = 1$ and it is not satisfied by the assignments in which $f(\boldsymbol{x}) = 0$. Given the definition of f_ϕ we can easily see that f_ϕ and f coincides. □

2. Satisfiability checking

The propositional satisfiability problem (often called SAT) is the problem of determining whether a set of sentences in Propositional Logic is satisfiable. The problem is significant both because the question of satisfiability is important in its own right and because many other questions in Propositional Logic can be reduced to that of propositional satisfiability. In practice, many automated reasoning problems in Propositional Logic are first reduced to satisfiability problems and then by using a satisfiability solver. Today, SAT solvers are commonly used in hardware design, software analysis, planning, mathematics, security analysis, and many other areas.

[1] For every set S, $|S|$ indicates the number of elements of S, aka, the *cardinality* of S

2.1. Truth tables.

EXAMPLE 1.2. *Let*

$$\Gamma = \{p \vee q, p \vee \neg q, \neg p \vee q, \neg p \vee \neg q \vee \neg r, \neg p \vee r\}$$

We want to determine whether Γ *is satisfiable. So, we build a truth table for this case.*

p	q	r	$p \vee q$	$p \vee \neg q$	$\neg p \vee q$	$\neg p \vee \neg q \vee \neg r$	$\neg p \vee r$	Γ
0	0	0	0	1	1	1	0	0
0	0	1	0	1	1	1	1	0
0	1	0	1	0	1	1	1	0
0	1	1	1	0	1	1	1	0
1	0	0	1	1	0	1	0	0
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	0	1	0

In a truth table there is one row for each possible truth assignment. For each truth assignment, each of the sentences in Γ *is evaluated. If any sentence evaluates to 0, then* Γ *as a whole is not satisfied by the truth assignment. If a satisfying truth assignment is found, then* Γ *is determined to be satisfiable. If no satisfying truth assignment is found, then* Γ *is unsatisfiable. In this example, every row ends with* Γ *not satisfied. So the truth table method concludes that* Γ *is unsatisfiable (not satisfiable).*

The truth table method is complete because every truth assignment is checked. However, the method is impractical for all but very small problem instances. In our example with 3 propositional variables, there are $2^3 = 8$ rows. For a problem instance with 10 propositional variables, there are $2^{10} = 1024$ rows - still quite small for a modern computer. But as the number of propositions grow, the number of rows quickly overwhelms even the fastest computers. A more efficient method is needed.

3. Normal forms

The methods to check satisfiability that are more efficient that truth tables require that the input (set of) formula(s) have a specific structure. The description of such a structure is also called *Normal Form*. A normal form is a pattern for the structure of a formulas. A formula is in a specific normal form if its structure matches the pattern of the normal form. An important property of normal forms is that every formula can be transformed in an equivalent formula which is in normal form. This property is important since the semantics of the formula should be preserved by the transformation in normal form. There are many possible normal forms, e.g., negated normal form, conjunctive normal form, disjunctive normal form, deterministic disjunctive normal form ..., each of which is suitable for solving a specific problem. In this chapter, we introduce two normal forms, namely: Negated Normal Form and Conjunctive Normal Form, which are normal forms suitable for checking satisfiability. Further normal forms will be introduced in successive chapter where we will consider other problems such as for instance model counting and weighted model counting.

Let us first introduce some terminology. A *literal* is either an atomic formula or the negation of an atomic formula. Examples of literals are p, q, $\neg p$, $\neg q$. A literal is positive if it is an antom; a literal is negative if it is the negation of an atom.

3.1. Negation Normal Form (NNF).

DEFINITION 1.2 (Negation Normal Form). *A formula is in* negation normal form *(NNF) if negation connective ($\neg$) occurs only in literals; or equivalently if the negation connective occurs only in front of atomic formulas.*

EXAMPLE 1.3. $p \wedge \neg q$ *is in NNF,* $\neg(p \wedge q)$ *is not in NNF.*

Any formula can be transformed into an equivalent formula in NNF. by pushing the negation operator inwards, applying the following transformations that preserves the semantics.

$$\begin{aligned} \neg\neg\phi &\Rightarrow \phi \\ \neg(\phi \wedge \psi) &\Rightarrow \neg\phi \vee \neg\psi \\ \neg(\phi \vee \psi) &\Rightarrow \neg\phi \wedge \neg\psi \\ \neg(\phi \rightarrow \psi) &\Rightarrow \phi \wedge \neg\psi \\ \neg(\phi \leftrightarrow \psi) &\Rightarrow (\phi \vee \psi) \wedge (\neg\phi \vee \neg\psi) \end{aligned} \tag{1}$$

PROPOSITION 1.3. *If ϕ' is obtained by applying any sequence of transofmration rules in* (1) *to ϕ, then ϕ is equivalent to ϕ'.*

PROOF. Recall that ϕ is equivalent to ϕ' if and only if for every ionterpretation $\mathcal{I}$ of the propositional variables in ϕ and ϕ', $\mathcal{I} \models \phi$ if and only if $\mathcal{I} \models \phi'$. Let us start with the first rule:

$$\mathcal{I} \models \neg\neg\phi \text{ iff } \mathcal{I} \not\models \neg\phi \text{ iff } \mathcal{I} \models \phi$$

$$\begin{aligned} \mathcal{I} \models \neg(\phi \wedge \psi) &\text{ iff } \mathcal{I} \not\models \phi \wedge \psi \text{ iff } \text{ either } \mathcal{I} \not\models \phi \text{ or } \mathcal{I} \not\models \psi \\ &\text{ iff } \text{ either } \mathcal{I} \models \neg\phi \text{ or } \mathcal{I} \models \neg\psi \text{ iff } \mathcal{I} \models \neg\phi \vee \neg\psi \end{aligned}$$

$$\begin{aligned} \mathcal{I} \models \neg(\phi \vee \psi) &\text{ iff } \mathcal{I} \not\models \phi \wedge \psi \text{ iff } \mathcal{I} \not\models \phi \text{ and } \mathcal{I} \not\models \psi \\ &\text{ iff } \mathcal{I} \models \neg\phi \text{ and } \mathcal{I} \models \neg\psi \text{ iff } \mathcal{I} \models \neg\phi \wedge \neg\psi \end{aligned}$$

$$\begin{aligned} \mathcal{I} \models \neg(\phi \rightarrow \psi) &\text{ iff } \mathcal{I} \not\models \phi \rightarrow \psi \text{ iff } \mathcal{I} \models \phi \text{ and } \mathcal{I} \not\models \psi \\ &\text{ iff } \mathcal{I} \models \phi \text{ and } \mathcal{I} \models \neg\psi \text{ iff } \mathcal{I} \models \phi \wedge \neg\psi \end{aligned}$$

$$\begin{aligned} \mathcal{I} \models \neg(\phi \leftrightarrow \psi) &\text{ iff } \mathcal{I} \not\models \phi \equiv \psi \\ &\text{ iff either } \mathcal{I} \models \phi \text{ and } \mathcal{I} \not\models \psi \text{ or } \mathcal{I} \not\models \phi \text{ and } \mathcal{I} \models \psi \\ &\text{ iff either } \mathcal{I} \models \phi \text{ and } \mathcal{I} \models \neg\psi \text{ or } \mathcal{I} \models \neg\phi \text{ and } \mathcal{I} \models \psi \\ &\text{ iff either } \mathcal{I} \models \phi \wedge \neg\psi \text{ or } \mathcal{I} \models \neg\phi \wedge \psi \\ &\text{ iff } \mathcal{I} \models (\phi \wedge \neg\psi) \vee (\neg\phi \wedge \psi) \\ &\text{ iff } \mathcal{I} \models (\phi \vee \psi) \wedge (\neg\phi \vee \neg\psi) \end{aligned}$$

□

Notice that the presence of the $\leftrightarrow$ connective in the formula makes its negated normal form to to exponentially increase in the size. For instance the NNF of

$$\neg(p \leftrightarrow \neg(q \leftrightarrow r))$$

is equal to

$$(\neg p \wedge (p \leftrightarrow r)) \vee (p \wedge ((q \wedge \neg r) \vee (\neg q \wedge r)))$$

One final observation about Negated Normal Form concerns the fact that the "negative information" present in a NNF formula is not encoded exclusively by negative literals. For instance, an implication of the form $p \to q$ encodes some negative information despite the absence of negative literals. Indeed, when q is evaluated to false, p should also be false. This can be also seen by the fact that $p \to q$ is equivalent to $\neg p \vee q$. For this reason NNF is usally combined with other normal forms which remove the $\to$ and $\leftrightarrow$ connective by restriting $\phi \to \psi$ as $\neg\phi \vee \psi$ and $\phi \leftrightarrow \psi$ as $(\neg\phi \vee \psi) \wedge (\phi \vee \neg\psi)$.

3.2. Conjunctive Normal Form (CNF). Conjunctive Normal Form (CNF) is one of the most common form taken in input by algorithm for satisfiability checking. CNF is more specific than NNF (therefore every formula in CNF is also in NNF) and uses only the connectives $\vee$, $\wedge$ and $\neg$. We introduce the notion of clause.

A *clause* is a formula of the form $l_1 \vee l_2 \vee \cdots \vee l_n$ where l_i are literals. An example of clause is

$$a \vee \neg b \vee c$$

An interpretation $\mathcal{I}$ satisfies a clause $l_1 \vee \cdots \vee l_n$ if *there is* a literal l_i that is satisfied by $\mathcal{I}$. We also admit clauses that are composed of zero literals, which is called *empty clause*. By applying the same criteria for satisfiability of a clause, we have that an interpretation *never* satisfies the empty clause. For this reason the empty clasue can be consistently denoted by the $\perp$ symbol, representing the proposition that is always false. The clauses that contains only one literal nare called *unit clauses*. An interpretation satisfies a unit clause if and only if it assign to true (resp. false) the positive (resp. negative) literal it contains. Empty clauses and unit clauses play an important role in the satisfiability checking procedure, this is why they received a special name.

Definition 1.3 (Conjunctive normal form). *A formula is in* conjunctive normal form *if if is the conjunction clauses.*

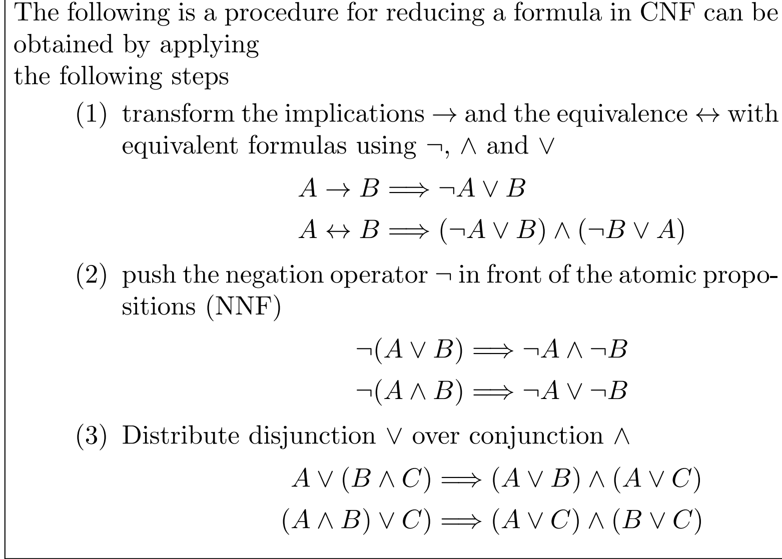
Example 1.4. *The formula*

$$(p \vee \neg q \vee r) \wedge (q \vee \neg r) \wedge (\neg p \vee \neg q) \wedge r$$

is in CNF; it is composed by the following four clauses:

$$C_1 = p \vee \neg q \vee r$$
$$C_2 = q \vee \neg r$$
$$C_3 = \neg p \vee \neg q$$
$$C_4 = r$$

that contains 3, 3, 2, and 1 literal respectively.



The following is a procedure for reducing a formula in CNF can be obtained by applying
the following steps

(1) transform the implications $\rightarrow$ and the equivalence $\leftrightarrow$ with equivalent formulas using $\neg$, $\wedge$ and $\vee$

$$A \rightarrow B \Longrightarrow \neg A \vee B$$
$$A \leftrightarrow B \Longrightarrow (\neg A \vee B) \wedge (\neg B \vee A)$$

(2) push the negation operator $\neg$ in front of the atomic propositions (NNF)

$$\neg(A \vee B) \Longrightarrow \neg A \wedge \neg B$$
$$\neg(A \wedge B) \Longrightarrow \neg A \vee \neg B$$

(3) Distribute disjunction $\vee$ over conjunction $\wedge$

$$A \vee (B \wedge C) \Longrightarrow (A \vee B) \wedge (A \vee C)$$
$$(A \wedge B) \vee C) \Longrightarrow (A \vee C) \wedge (B \vee C)$$

FIGURE 1. CNF reduction rules.

Due to the fact that $\phi \vee \phi$ and $\phi \wedge \phi$ are equivalent to ϕ (identical operand) and the fact that $\phi \vee \psi$ and $\phi \wedge \psi$ are equivalent to $\psi \vee \phi$ $\psi \wedge \phi$ (commutativity) we can consider a clause as a *set of literals*, and a CNF formula as a *set of sets of literals*. For instance the previous example can be seen as the set that contains four sets (clauses)

$$\{\{p, \neg q, r\}, \{q, \neg r\}, \{\neg p, \neg q\}, \{r\}\}$$

PROPOSITION 1.4. *Every formula can be rewritten in an equivalent formula in CNF.*

As for the case of NNF we provide a set of rewriting rules, shown in Figure 1 that preserves the semantics (i.e., a formula is rewritten in an equivalent formula). To reduce a formula in CNF we therefore define a recursive algorithm CNF that computes the CNF of a formula ϕ by, first recursively compute the CNF of the sub-formulas of ϕ and then combine the results

EXAMPLE 1.5. *Let us transform the following formula in CNF*

$$\begin{aligned}
(p \rightarrow q) \wedge \neg q \rightarrow \neg p &\Longrightarrow ((p \rightarrow q) \wedge \neg q) \rightarrow \neg p && \textit{explicit the scope of connectives} \\
&\Longrightarrow \neg((\neg p \vee q) \wedge \neg q) \vee \neg p && \textit{rewrite} \rightarrow \textit{in terms of} \neg \textit{and} \vee \\
&\Longrightarrow ((p \wedge \neg q) \vee q) \vee \neg p && \textit{move} \neg \textit{in front of atomic formulas} \\
&\Longrightarrow ((p \vee \neg q) \wedge (\neg q \vee q) \vee \neg p && \textit{distribute} \vee \textit{over} \wedge \\
&\Longrightarrow ((p \vee \neg q \vee \neg p) \wedge (\neg q \vee q \vee \neg p) && \textit{distribute} \vee \textit{over} \wedge
\end{aligned}$$

PROPOSITION 1.5. *CNF terminates for every input ϕ.*

PROOF. Notice that each rules (1), (2) and (3) can be applied only a finite number of times. □

After proving termination we also have to show that the transformation in CNF do not change the semantics of the formula, i.e., CNF(ϕ) and ϕ are equivalent.

PROPOSITION 1.6. *ϕ is equivalent to CNF(ϕ).*

PROOF. This is a consequence of the fact that every transformation applied by the procedure for reduction to CNF tranform a formula into an equivalent formula. □

An interpretation $\mathcal{I}$ satisfies a formula in CNF, or equivalently, a set of clauses $\{C_1, \ldots, C_n\}$, if $\mathcal{I}$ satisfies at least on literal $l_{ij} \in C_i$ for every clause C_i. This means that to check satisfiability of a set of clauses we have to find a truth assignment such that for every clauses it select one literal negative and assign 0 or 1 to the corresponding propositional variables depending on the fact that the literal is negative or positive.

EXAMPLE 1.6. *Consider the previous set of clauses*

$$\{\{p, \neg q, r\}, \{q, \neg r\}, \{\neg p, \neg q\}, \{r\}\} \tag{2}$$

An interpretation that satisfies (2) *should satisfy at least one literal per clause. Since the last clause contains only one literal (i.e. r) there is no other choice than satisfying it by setting $\mathcal{I}(r)$ = True. This choice would also be a good one for the first clause, since it contains also r which is satisfied by $\mathcal{I}$, therefore we have to take care only of the two remaining clauses:*

$$\{q, \neg r\}, \{\neg p, \neg q\}$$

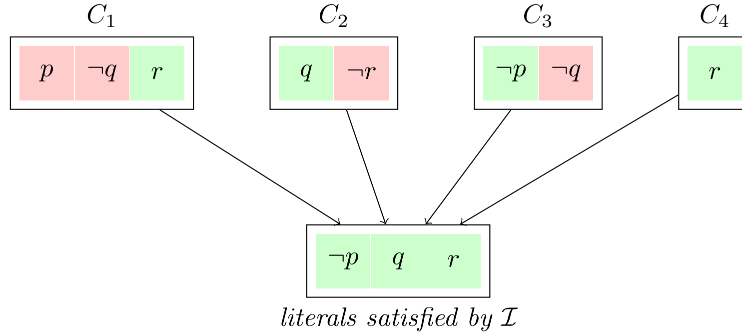
Considering the first of the above two clauses, one can see that the literal $\neg r$ is already falsified by our previous (obliged) choice of $\mathcal{I}(r)$ = True. Therefore, to satisfy the first clause, the only choice is to set $\mathcal{I}(q)$ = True. We ore now left with one last clause:

$$\{\neg p, \neg q\}$$

Since $\mathcal{I}(q)$ = True, the second literal cannot be satisfied; therefore we should satisfy $\neg p$ by setting $\mathcal{I}(p)$ = False. In conclusion, we have the interpretation

$$\mathcal{I}(p) = \mathit{False} \qquad \mathcal{I}(q) = \mathit{True} \qquad \mathcal{I}(r) = \mathit{True}$$

A graphical representation of the interpretation that we have found is the following:



literals satisfied by $\mathcal{I}$

3.3. Tseytin Transformation. Checking satisfiability/validity of a formula in CNF is easier. But there is a price in terms of the size of formulas. Indeed due the restriction imposed by CNF, representing some complex proposition in a CNF form can require an exponentially larger formula than by representing it with the full propositional language. As an intuitive example consider the formula $p \leftrightarrow q$ which

in CNF double its size becoming $(\neg p \vee q) \wedge (\neg q \vee p)$. Another example, happens when the formul ais a disjunction of conjunct (instead of a conjuction of disjunct) e.g., the formula $(a \wedge b) \vee (c \wedge d)$ expands into $(a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d)$ which is twice the size of the original formula. When these patterns are nested, you can easily immagine that the explosion is of the order of 2^n where n iis the maximum nesting. Let us consider an example.

EXAMPLE 1.7 (Exponential explosion). *The CNF of the formula* $p \leftrightarrow (q \leftrightarrow (r \leftrightarrow s))$ *is composed by the following clauses*

$$\begin{array}{cccc} (\neg r, \neg s, \neg q, p) & (r, s, \neg q, p) & (\neg s, r, q, p) & (\neg r, s, q, p) \\ (\neg r, \neg s, q, \neg p) & (r, s, q, \neg p) & (\neg s, r, \neg q, \neg p) & (\neg r, s, \neg q, \neg p) \end{array}$$

If we add an extra equivalence and compute the CNF of

$$p \leftrightarrow (q \leftrightarrow (r \leftrightarrow (s \leftrightarrow t)))$$

we obtain twice the number of clauses

$$\begin{array}{cccc} (\neg s, \neg t, \neg r, \neg q, p) & (s, t, \neg r, \neg q, p) & (\neg t, s, r, \neg q, p) & (\neg s, t, r, \neg q, p) \\ (\neg s, \neg t, r, q, p) & (s, t, r, q, p) & (\neg t, s, \neg r, q, p) & (\neg s, t, \neg r, q, p) \\ (\neg s, \neg t, \neg r, q, \neg p) & (s, t, \neg r, q, \neg p) & (\neg t, s, r, q, \neg p) & (\neg s, t, r, q, \neg p) \\ (\neg s, \neg t, r, \neg q, \neg p) & (s, t, r, \neg q, \neg p) & (\neg t, s, \neg r, \neg q, \neg p) & (\neg s, t, \neg r, \neg q, \neg p) \end{array}$$

The exponential explosion of the CNF transformation can be contrasted by avoiding to do multiple time the same CNF expansion. To this aim, G. Tseytin in Tseytin 1966 proposed the following procedure:

(1) for a subformula ψ of ϕ which is not a literal, introducing new propositional variables p;
(2) (ii) replace all the occurrences of ψ with the p, and
(3) add a new conjunction stating that p and ϕ are equivlent i.e $\psi \leftrightarrow p$.

In short we apply this transformation:

$$\phi \Rightarrow \phi[\psi/p] \wedge p \leftrightarrow \psi \tag{3}$$

where p is a "fresh" propositional variable, i.e., a propositional variable that do not appear in ϕ, and the notation $\phi[\psi/p]$ indicates the formula ϕ obtained by replacing each occurrences of the subformula ψ with the propositional variable p. This transformation has the advantage that the exponential blow up of the "CNF-ization" of ψ happens only onece instead the number of occurrences of ψ in ϕ.

The transformation (3) however does not preserve the semantics of the formula, i.e., the resulting formula is *not equivalent* to the original formula. For instance the formula $(a \rightarrow b) \rightarrow (c \rightarrow (a \rightarrow b))$ is not equivalent to the formula $(p \rightarrow (c \rightarrow p)) \wedge (p \leftrightarrow (a \rightarrow b))$, obtained by replacing the subformula $a \rightarrow b$ with p. Indeed there are interpretations of the first formulas which do not satisfy the second. For instance the interpretation with $\mathcal{I}(p) = \text{true}$ and $\mathcal{I}(a) = \text{True}$ and $\mathcal{I}(b) = \text{False}$ satisfies the first formula but not the second. This does not constitute a serious problem, since we are interesting in checking satisfiability. For this task it is sufficient that the original formula is satisfiable if and only if the resulting formula is also satisfiable.

We therefore inroduce a weaker notion than equvalence, which is enough for checking satisfiability.

Definition 1.4 (Equi-satisfiable formulas). *Two formulas* ϕ *and* ϕ' *are* equi-satisfiable *if,* ϕ *is satisfiable if and only if* ϕ' *is satisfiable*

Equi-satisfiability is weaker than equivalence; indeed it is easy to find pairs of equi-satisfiable formulas that are not equivalent. For instance any pair of atomic formulas p and q are equisatisfiable (indeed they are both satisfiable) but they are not equivalent, Since they are satisfied by different interpretations. Let us now prove that the transformation 3 preserves satisfiability. One can see the difference between equivalent and equi-satisfiability by looking at their formal definition:

ϕ and ψ are equivalent	$\forall \mathcal{I}.\ \mathcal{I} \models \phi \Leftrightarrow \mathcal{I} \models \psi$
ϕ and ψ are equi-satisfiable	$\exists \mathcal{I}.\ \mathcal{I} \models \phi \Leftrightarrow \exists \mathcal{J}.\ \mathcal{J} \models \psi$

Proposition 1.7. *For every formula* ϕ *and avery propositional variable* p *not occurring in* ϕ*,* ϕ *and* $\phi[\psi/p] \wedge p \leftrightarrow \psi$ *are equisatisfiable.*

Proof. Suppose that ϕ is satisfiable by an interpretation $\mathcal{I}$ let $\mathcal{I}'$ be the interpretation obtained from $\mathcal{I}$ by setting

$$\mathcal{I}'(p) = \begin{cases} \text{True} & \text{if } \mathcal{I} \models \psi \\ \text{False} & \text{Otherwise} \end{cases}$$

By construction $\mathcal{I}' \models p \leftrightarrow \psi$, To show that $\mathcal{I}' \models \phi[\psi/p]$ we have to proceed by induction.

- **Base case:** ϕ is equal to ψ. Then $\mathcal{I} \models \phi$ implies that $\mathcal{I}' \models p$ and since $\phi[\psi/p]$ is equal to p, we have that $\mathcal{I}' \models \phi[\psi/p]$. If instead ψ is not a subformula of ϕ then $\phi[\psi/p] = \phi$ and since ϕ does not contain p, it has the same truth value w.r.t., $\mathcal{I}'$ and $\mathcal{I}$, and therefore $\mathcal{I}' \models \phi[\psi/p]$.
- **Step case:** Suppose that ϕ is $\phi_1 \vee \phi_2$, We have that $\mathcal{I} \models \phi_i$ for some $i = 1, 2$. By the induction hypothesis we have that $\mathcal{I}' \models \phi_i[\psi/p]$ Since $\phi[\psi/p]$ is equal to $\phi_1[\psi/p] \vee \phi_2[\psi/p]$, we can conclude that $\mathcal{I}' \models \phi[\psi/p]$.
- ... Do the other cases by exercize.

□

The Tseytin transformation introduced in Tseytin 1966 applies the (3) rule in a sistematic way, This correspond to the following procedure that can be applied to any formula ϕ (not necessariliy in CNF).

(1) let $\psi_1, \ldots, \psi_n$ all the subformulas of ϕ, including ϕ itself and excluding those that are literals[2]: make sure that if ψ_i is a subformula of ψ_j then $j \geq i$;
(2) let $x_1, \ldots, x_n$ a set of propositional variables not occurring in ϕ.
(3) for $i = 1, \ldots, n$ apply the following transformation:

$$\phi \Rightarrow \phi[\psi_i/x_i] \wedge (x_i \leftrightarrow \psi_i)$$

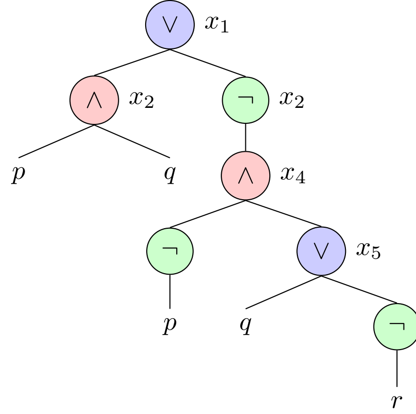
(4) apply CNF to the resulting formula.

Example 1.8. *Let us consider the formula*

$$(p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$$

With the following parse tree and subformulas

[2] In the original formulation also negative literal were included.



The Tseytin's transformation introduces 5 additional variable $x_1, \dots, x_5$ since the formula has 5 subformuls which are not literals. In the following table, we provide the definition of each x_i in terms of it's direct subformulas, and the corresponding transformation in CNF.

$$
\begin{array}{ll}
x_1 \leftrightarrow x_2 \vee x_3 & \leftrightarrow \{\neg x_1, x_2, x_3\}, \{\neg x_2, x_1\}, \{\neg x_3, x_1\} \\
x_2 \leftrightarrow p \wedge q & \leftrightarrow \{\neg x_2, p\}, \{\neg x_2, q\}, \{\neg p, \neg q, x_2\} \\
x_3 \leftrightarrow \neg x_4 & \leftrightarrow \{\neg x_3, \neg x_4\}, \{x_3, x_4\} \\
x_4 \leftrightarrow x_5 \wedge \neg p & \leftrightarrow \{\neg x_4, x_5\}, \{\neg x_5, x_4\} \\
x_5 \leftrightarrow q \vee \neg r & \leftrightarrow \{\neg x_5, q, \neg r\}, \{\neg q, x_5\}, \{r, x_5\}
\end{array}
$$

To check if the initial formula is satisfiable it is sufficent to check if x_1 (which is the auxiliary variable that encodes the formula itself) and all the CNF provided above are satisfiable.

Tseytin's tranformation transform a formula ϕ into a set of clauses that contains at most three literals as all the formula that are given in input to CNF are of the form $x_i \leftrightarrow (l_1 \circ l_2)$ or $x_i \leftrightarrow \neg x_j$, where $\circ$ is some binary connective in $\{\wedge, \vee, \rightarrow, \leftrightarrow\}$. The connective that generates the largest number of clauses is $\leftrightarrow$, and they are 4. Indeed, CNF$(a \leftrightarrow (b \leftrightarrow c))$ contains four clauses, namely: $\{a, b, c\}$, $\{a, \neg b, \neg c\}$, $\{\neg a, b, \neg c\}$ and $\{\neg a, \neg b, c\}$. This means that if a formula ϕ contains n connectives its Tseytin tranformed contains at most $4 \times n$ clauses each of which contains at most 3 literals. Therefore the size of the CNF obtained with the Tseytin tranformation is linear w.r.t., the number of connectives of the original formula. This is clearly a good news for the deciding satisfiability but this reduction comes with the price of introducing n new propositional variable, for which the satisfiability d decision procedure has to produce a truth assignment.

4. Davis-Putnam-Logemann-Loveland Procedure (DPLL)

The Davis-Putnam-Logemann-Loveland (DPLL) algorithm Davis and Putnam 1960; Davis, Logemann, and Loveland 1962 is a procedure that combines search and deduction to decide satisfiability of CNF formulas. This algorithm underlies most modern SAT solvers. While the basic procedure itself has been developed in the 60's, practical DPLL-based SAT solvers only started to appear from the mid 1990s as a result of enhancements such as clause learning, non-chronological

backtracking, branching heuristics, restart strategies, and lazy data structures. The DPLL algorithm is based around backtrack search for a satisfying valuation.

Before introducing the DPLL algorithm, we need to provide some notation and a key subruting exploited by DPLL, called Unit Propagation that deals with the unit clauses. A *partial interpretation* is a truth assignment for a subset of the propositional variables of a formula. As we have seen in the section of model checking, partial assignments in some cases can fully determine the truth value of some formulas. For instance if p is assigned to True by a partial interpretation then any formula of the form $p \vee \phi$ will be evaluated to true, independently from the truth value of ϕ. A partial evaluation can be seen as a set of literals. We will use the same notation for interpretations and partial interpretations. $p \in \mathcal{I}$ means that $\mathcal{I}(p) = \text{True}$ $\neg p \in \mathcal{I}$ means that $\mathcal{I}(p) = \text{False}$. We also use the notation $\bar{l}$ to denote the opposite of the literal l. I.e., $\bar{p}$ is $\neg p$ and $\overline{\neg p}$ is p. Similar definition can be defined also in sets of literals C. $\bar{C} = \{\bar{l} \mid l \in C\}$.

Given a partial assignment $\mathcal{I}$, we can attempt to evaluate a clause C.

- A clause C is true under $\mathcal{I}$ if one of its literals is true, i.e if $\mathcal{I} \cap C \neq \emptyset$
- A clause C is false under $\mathcal{I}$ if all its literal are set to false by $\mathcal{I}$, i.e. $C \subseteq \bar{\mathcal{I}}$.
- therwise C is undefined (or "unresolved") in $\mathcal{I}$.

When a clause is unresolved in an assignment $\mathcal{I}$ we can simplify it by evaluating the literals that are assigned by $\mathcal{I}$, For a clause C and a literal l the semplification of C w.r.t., l, denoted by $C|_l$ denotes the clause obtained by evaluating C w.r.t., the partial evaluation that assigns the literal l to true (i.e. $l \in \mathcal{I}$). $C|_l$ is obtained from C by the following two operations:

- removing all the clauses that contains l;
- remove $\bar{l}$ (if present) from C,

For any CNF formula ϕ, $\phi|_l = \{C|_l \mid C \in \phi\}$. For a set of literals $\{l_1, \ldots, l_n\}$, $C_{l_1,\ldots,l_n}$ is equal to $(\ldots(C|_{l_1})|_{l_2}\ldots)|_{l_n}$ Notice that the order and the repetitions does not affect the result , and therefore we can use the notation $\phi|_{\mathcal{I}}$ where $\mathcal{I}$ is a set of literal for denotign $\phi|_{l_1,\ldots,l_n}$, where $\mathcal{I} = \{l_1, \ldots, l_n\}$.

The subruting that is used by DPLL is called unit propagation and is applied when a CNF formula contains some unit clause. If ϕ contains unit clause $\{l\}$ then, to satisfy ϕ we have to satisfy $\{l\}$ and therefore the literal l must be evaluated to True. As a consequence ϕ can be simplified using the procedure called UnitPropagation. The procedure is shown in algorithm **??**.

Algorithm 1 UnitPropagation(ϕ : CNF, $\mathcal{I}$: Partial assignment)

```
1: while φ contains a unit clause {l} do
2:     I, φ ← I ∪ {l}, φ|_l
3:     if {} ∈ φ then
4:         return I, φ
5:     end if
6: end while
7: return I, φ
```

The basic DPLL algorithm is shown in Figure 1. DPLL algorithm incrementally build a a partial interpretation that satisfies the input CNF-formula by depth-first search. At any time the state of the algorithm is a pair $(\mathcal{I}, \phi)$, where $\mathcal{I}$ is a partial

truth assignment and ϕ is the set of clauses that are undecided by $\mathcal{I}$. This means that $\mathcal{I}$ must be extended in order to satisfy all the remaining clauses which are in ϕ.

Algorithm 2 DPLL(ϕ : CNF, $\mathcal{I}$: Partial assignment)

1: $\mathcal{I}, \phi \leftarrow$ UNITPROPAGATION($\mathcal{I}, \phi$)
2: **if** $\{\} \in \phi$ **then**
3: **return** UNSATISFIABLE
4: **end if**
5: **if** $\phi = \{\}$ **then**
6: **return** $\mathcal{I}$
7: **else**
8: select a l from some clause $C \in \phi$
9: $\mathcal{I} = $ DPLL($\phi|_l, \mathcal{I} \cup \{l\}$)
10: **if** $\mathcal{I} \neq$ UNSATISFIABLE **then**
11: **return** $\mathcal{I}$
12: **else**
13: $\mathcal{I} \leftarrow$ DPLL($\phi|_{\bar{l}}, \mathcal{I} \cup \{\bar{l}\}$)
14: **return** $\mathcal{I}$
15: **end if**
16: **end if**

EXAMPLE 1.9. *Let us consider a concrete example and look at how DPLL behaves on the following CNF ϕ.*

$$\{A, B\} \tag{4}$$
$$\{B, C\} \tag{5}$$
$$\{\neg A, \neg X, Y\} \tag{6}$$
$$\{\neg A, X, Z\} \tag{7}$$
$$\{\neg A, \neg Y, Z\} \tag{8}$$
$$\{\neg A, X, \neg Z\} \tag{9}$$
$$\{\neg A, \neg Y, \neg Z\} \tag{10}$$

Figure 2 shows the execution trace of DPLL algorithm on the set of clauses (4)–(10)*. The input set of clauses do not contain a unit clauses therefore unit propagation is not applied, furthermore $\mathcal{I}$ is not Unsatisfiable and and the set of clauses ϕ are not empty. Therefore DPLL goes to line 2 and select the literal A. Then it recursively apply DPLL to $\phi|_A$ and $\mathcal{I} = \{A\}$. The recursive calls proceeds similarly by selecting first the literal B and then X; At this point DPLL is called with input $\phi|_{A,B,X}$ and $\mathcal{I} = \{A, B, x\}$. Since $\phi_{A,B,X}$ contains a unit clause $\{Y\}$ then Y then the unit propagation is called. This procedure executes two steps by first propagating Y and then propagating Y, returning the set of clauses that containt the empty clause. At this point DPLL returns* UNSATISFIABLE *and backtrack (shown in dashed lines) at the most recent choice, which is on the literal X. Therefore, it analize the case of the negative literal $\neg X$, calling DPLL on $\phi_{A,B,\neg X}$ and $\mathcal{I} = \{A, B, \neg X\}$. But also in this case DPLL after calling unit propagation returns* UNSATISFIABLE, *Therfore, DPLL backtrack to the upper literal that has been choosen which is B*

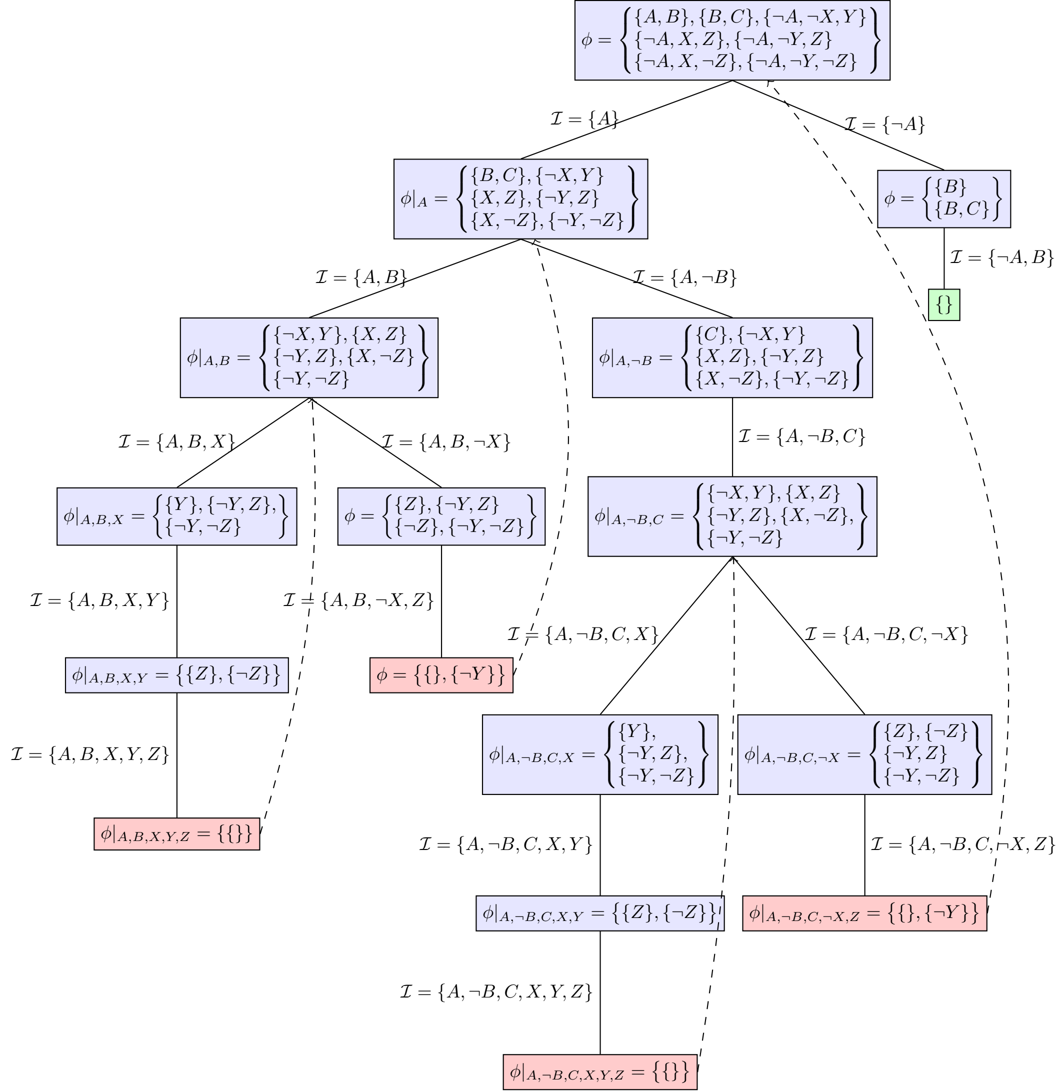


Figure 2. The execution trace of DPLL algorithm on the set of clauses (4)–(10).

.

and explores the case with $\neg B$*. As one can see from the trace also in this case all the branches leads to a set of unsatisfiable clauses (empty clause). Therfore DPLL backtrack to the initial choiche, which was on the literal* A*, and attemst with the assignment* $\mathcal{I} = \{\neg A\}$*. This leads after a unit propoatation to the emtpy set of clauses, and tDPLL returns the partial assignment* $\{\neg A, B\}$*, and therefore the initial set of clauses are satisfiable, by all the assignments that agree with the partial assignment returned by DPLL.*

5. Sat Solvers

Most of of the satate-of-the-art implementation of sat solvers are available through Python interface. In this section we summarize the main features offered by `PySAT`. Before introducing the library, we introduce a format for representation of CNF, called `DIMACS`.

5.1. `DIMACS` CNF.

The `DIMACS` CNF format is a textual representation of a formula in conjunctive normal form. A formula in conjunctive normal form is a conjunction (logical and) of a set of clauses. Each clause is a disjunction (logical or) of a set of literals. A literal is a variable or a negation of a variable. `DIMACS` CNF uses positive integers to represent variables and their negation to represent the corresponding negated variable. This convention is also used for all textual input and output in Varisat.

There are several variations and extensions of the `DIMACS` CNF format. Varisat tries to accept any variation commonly found. Currently no extensions are supported.

`DIMACS` CNF is a textual format. Any line that begins with the character c is considered a comment. Some other parsers require comments to start with c and/or support comments only at the beginning of a file. Varisat supports them anywhere in the file.

A `DIMACS` file begins with a header line of the form p cnf ¡variables¿ ¡clauses¿. Where ¡variables¿ and ¡clauses¿ are replaced with decimal numbers indicating the number of variables and clauses in the formula.

Varisat does not require a header line. If it is missing, it will infer the number of clauses and variables. If a header line is present, though, the formula must have the exact number of clauses and may not use variables represented by a number larger than indicated.

Following the header line are the clauses of the formula. The clauses are encoded as a sequence of decimal numbers separated by spaces and newlines. For each clause the contained literals are listed followed by a 0. Usually each clause is listed on a separate line, using spaces between each of the literals and the final zero. Sometimes long clauses use multiple lines. Varisat will accept any combination of spaces and newlines as separators, including multiple clauses on the same line.

EXAMPLE 1.10. *As an example the formula* $(x \vee y \vee \neg z) \wedge (\neg y \vee z)$ *could be encoded as this:*

```
p cnf 3 2
1 2 -3 0
-2 3 0
```

The first line "`p cnf 3 2`*" states that this file specifies a* `p`*roblem in* `cnf` *with* `3` *propositional variables and* `2` *clauses. The other two lines specify the clauses (every clause ends with a* `0`*).*

5.2. Problem solving with Sat Solvers.

In this section we describe how to compute a solution of a problem that you have specified in terms of a set of propositional formula Γ that uses the set of propositional variables $\mathcal{P}$. Usually $\mathcal{P}$ contains propositional variable with some "human readable" name. For instance we can use the the propositional variable $\mathsf{happy(John)}$ to remind us that it represents the proposition that John is happy.

5.2.1. *Encoding and Decoding propositinal variables.* The first operation that you have to perfomr is to define an function (encoding) that maps each $p \in \mathcal{P}$ into a natural number. i.e, the function

$$\texttt{Encode} : \mathcal{P} \longrightarrow [n]$$

where $[n] = \{1, 2, \ldots, n\}$ and n is the number of peropositional variables which are present in $\mathcal{P}$. The function `Encode` cannot associate the same integer to two different propositional variables, i.e., `Encode` is injective. This property allows us to define the `Decode` function from $[n]$ to $\mathcal{P}$ as the inverse of `Encode`.

$$\texttt{Decode} : [n] \longrightarrow \mathcal{P}$$

is such that $\texttt{Decode}(i) = \texttt{Encode}^{-1}(i)$.

5.2.2. *Generating the* `DIMACS` *code.* To generate the `DIMACS` code we have first to generate the first line, which is

$$\texttt{p CNF } |\mathcal{P}| \; |\Gamma|$$

that means that this is a cnf problem on $|\mathcal{P}|$ proposition and Γ clauses. Then for every clause $\gamma = \{l_1, \ldots, l_k\} \in \Gamma$ we cenerate the line $\texttt{Encode}(p)$ if l_i is a positive literal p and $-\texttt{Encode}(p)$ if l_i is a negative literal $\neg p$.

5.2.3. *Decoding solution.* We then call the sat solver on the encoding that we have generated. The sat solver can return either `UNSATISFIABLE` or `SATISFIABLE`. In the latter case we can ask the solver to return a mnodel. The sat solver return a model codified in a sequence of positive and negative integers.

$$r = [\pm 1, \pm 2, \ldots, \pm n]$$

where n is the total number of propositional variables. The list encodes the following interpretation: for every $p \in \mathcal{P}$

$$\mathcal{I}_r(p) = \begin{cases} \mathsf{True} & \text{if } \texttt{Encode}(p) \in r \\ \mathsf{False} & \text{if } -\texttt{Encode}(p) \in r \end{cases}$$

5.2.4. *Additional solutions.* If we want to obtain additional models we have to add to the initial set of clauses that fact that we want a model different from $\mathcal{I}_r$. A model of Γ is different from $\mathcal{I}_r$ if at least one proposition takes a truth value different from the one assigned by $\mathcal{I}_r$. We can therefore add the clause

$$\bigvee_{\mathcal{I}_r(p)=\mathsf{True}} \neg p \vee \bigvee_{\mathcal{I}_r(p)=\mathsf{False}} p$$

which can be codified in the sequence $-r = [-i \mid i \in r]$.

5.3. PySAT. see website `https://pysathq.github.io/`

6. Exercises

Exercise 1:

Explain why $\phi \models \psi$ holds if and only if $\{\phi, \neg\psi\}$ is not satisfiable.

Exercise 2:

How many conjunctive normal forms formulas can you build with n propositional variables? Remember that a conjunctive normal form can be seen as a set of set of literals.

Solution A CNF formula is a set of clauses. A clause is a set of literals. With n propositional variables we can build $2n$ literals, i.e., the positive literals, which coincide with the propositional variables, and the negative literals, which are the negation of propositional variables. A clause is any set of literals, Therefore we can build 2^{2n} clauses. A CNF formula is a set of clauses, therefore we can build $2^{2^{2n}}$ CNF formulas. □

Exercise 3:

Transform the following formula in CNF

$$((\neg((p \to q) \land (p \lor q \to r)) \to (p \to r)))$$

Exercise 4:

Explain the difference between the following two facts: "ϕ and ψ are equivalent"and "ϕ and ψ are equisatisfiable"

Solution ϕ is equivalent to ψ if and only if for all the interpretations $\mathcal{I}$, $\mathcal{I} \models \phi$ if and onlyif $\mathcal{I} \models \psi$.

ϕ and ψ are equisatisfiable if ϕ is satisfiable if and only if ψ is satisfiable. □

Exercise 5:

Consider the following two formulas:

$$\phi = P \to ((Q \to R) \land (Q \lor R))$$
$$\psi = (\neg P \to Q) \to R$$

Does ϕ logically follow from ψ? or viceversa? Prove it via truth tables.

Solution Let us compute the truth tables for both formulas:

	P	Q	R	P	→	((Q	→	R)	∧	(Q	∨	R))	(¬	P	→	Q)	→	R
1	T	T	T	T	T	T	T	T	T	T	T	T	F	T	T	T	T	T
2	T	T	F	T	F	T	F	F	F	T	T	F	F	T	T	T	F	F
3	T	F	T	T	T	F	T	T	T	F	T	T	F	T	T	F	T	T
4	T	F	F	T	F	F	T	F	F	F	F	F	F	T	T	F	F	F
5	F	T	T	F	T	T	T	T	T	T	T	T	T	F	T	T	T	T
6	F	T	F	F	T	T	F	F	F	T	T	F	T	F	T	T	F	F
7	F	F	T	F	T	F	T	T	T	F	T	T	T	F	F	F	T	T
8	F	F	F	F	T	F	T	F	F	F	F	F	T	F	F	F	T	F

Notice that in all the cases in which ϕ is true ϕ is also true, this implies that ϕ logically follows from ψ. On the other hand, since there is an assignment for which ϕ is true but ψ is false (assignment number 6) ψ is not a logical consequence of ϕ. □

Exercise 6:

Convert the following formula

$$p \lor q \to p \land \neg r$$

into an equisatisfiable CNF formula using the Tseitin's Transformation.

Exercise 7:

Check the following facts via DPLL (in the exam there could be one of this).

(1) $\models (p \to q) \land \neg q \to \neg p$
(2) $\models (p \to q) \to (p \to \neg q)$
(3) $\models (p \lor q \to r) \lor p \lor q$
(4) $\models (p \lor q) \land (p \to r \land q) \land (q \to \neg r \land p)$
(5) $\models (p \to (q \to r)) \to ((p \to q) \to (p \to r))$
(6) $\models (p \lor q) \land (\neg q \land \neg p)$
(7) $\models (\neg p \to q) \lor ((p \land \neg r) \equiv q)$
(8) $\models (p \equiv q) \lor (p \equiv \neg q)$
(9) $\models (p \to (q \lor r)) \lor (r \to \neg p)$
(10) $\models \neg(p \equiv q) \lor \neg(p \equiv \neg q)$
(11) $\models (p \equiv q) \lor (p \equiv \neg q)$

SolutionThe solution of some cases are the following:

(1) The expression $\models (p \to q) \land \neg q \to \neg p$ means that $(p \to q) \land \neg q \to \neg p$ is valid. To check the validity of a formula ϕ with DPLL we have to check (un)satisfiability of the negated of ϕ i.e. $\neg\phi$. Since DPLL requires CNF we first have to transform in CNF $\neg\phi$.

$$\begin{aligned}
\text{CNF}(\neg((p \to q) \land \neg q \to \neg p)) &= \text{CNF}((p \to q) \land \neg q) \land \text{CNF}(\neg\neg p)) \\
&= \text{CNF}((p \to q)) \land \text{CNF}(\neg q) \land \text{CNF}(p)) \\
&= \{\neg p, q\}, \{\neg q\}, \{p\}
\end{aligned}$$

Now we apply DPLL to $\{\neg p, q\}, \{\neg q\}, \{p\}$.

(a) since $\{\neg p, q\}, \{\neg q\}, \{p\}$ contains unit clauses, we apply UNITPROPAGATION.

$\{\{\neg p, q\}, \{\neg q\}, \{p\}\}$	$\mathcal{I} = \{\}$	Unit propagation on $\{p\}$
$\{\{\neg p, q\}, \{\neg q\}\}\|_p$	$\mathcal{I} = \{p\}$	
$\{\{q\}\{\neg q\}\}$	$\mathcal{I} = \{p\}$	Unit propagation on $\{q\}$
$\{\{\neg q\}\}\|_q$	$\mathcal{I} = \{p, q\}$	
$\{\{\}\}$	$\mathcal{I} = \{p, q\}$	We have obtained the empty clause, and we return UNSAT

Since we have obtained the empty clauses we return UNSAT. Since $\neg((p \to q) \land \neg q \to \neg p)$ is not satisfiable, it must be that $(p \to q) \land \neg q \to \neg p$ is valid.

(2) We first compute the CNF of the negation of the formula.

$$\begin{aligned}
\text{CNF}(\neg((p \to q) \to (p \to \neg q))) &= \text{CNF}(p \to q) \land \text{CNF}(\neg(p \to \neg q)) \\
&= \text{CNF}(p \to q) \land \text{CNF}(p) \land \text{CNF}(\neg\neg q) \\
&= \{\neg p, q\}, \{p\}, \{q\}
\end{aligned}$$

We can now apply DPLL

$\{\{\neg p, q\}, \{p\}, \{q\}\}$	$\mathcal{I} = \{\}$	Unit propagation on $\{q\}$
$\{\{\neg p, q\}\}\|_q, \{p\}\|_q \{q\}\|_q$	$\mathcal{I} = \{q\}$	
$\{\{p\}\}$	$\mathcal{I} = \{q\}$	Unit propagation on $\{p\}$
$\{\{p\}\}\|_p$	$\mathcal{I} = \{q, p\}$	
$\{\}$	$\mathcal{I} = \{q, p\}$	

Since we obtaine the empty set of clauses (all the clauses are eliminated), then we have that the set of input clauses are satisfiable, I.e., that $\neg((p \rightarrow q) \rightarrow (p \rightarrow \neg q))$ is satisfiable, and therefore $(p \rightarrow q) \rightarrow (p \rightarrow \neg q)$ is not valid. A counter-model, i.e. a model that falsifies it, is returned by the DPLL and it is equal to $\mathcal{I}(p) = True$ and $\mathcal{I}(q) = True$.

6. Let's compute the CNF of $\neg((p \vee q) \wedge (\neg q \wedge \neg p))$

$$
\begin{aligned}
\text{CNF}(\neg((p \vee q) \wedge (\neg q \wedge \neg p))) &= \text{CNF}(\neg(p \vee q)) \otimes \text{CNF}(\neg(\neg q \wedge \neg p)) \\
&= (\text{CNF}(\neg p) \wedge \text{CNF}(\neg q)) \otimes \text{CNF}(q \vee p) \\
&= (\neg p \wedge \neg q) \otimes (q \vee p) \\
&= (\neg p \vee q \vee p) \wedge (\neg q \vee q \vee p) \\
&= \{\neg p, q, p\}, \{\neg q, q, p\}
\end{aligned}
$$

We can now apply DPLL: Since there is no unit clauses, we skip that block and select a literal

$\{\{\neg p, q, p\}, \{\neg q, q, p\}\}$	$\mathcal{I} = \{\}$	select the literal p
$\{\{\neg p, q, p\}, \{\neg q, q, p\}\}\|_p$	$\mathcal{I} = \{p\}$	
$\{\}$	$\mathcal{I} = \{p\}$	We have eliminated all the clauses therefore we return SAT

Since the negated of the formula is satisfiable, then the initial formula cannot be valid.

10. We compute the CNF of the negated formula

$$
\begin{aligned}
\text{CNF}(\neg(\neg(p \equiv q) \vee \neg(p \equiv \neg q))) &= \text{CNF}((p \equiv q) \wedge (p \equiv \neg q)) \\
&= \text{CNF}((p \equiv q)) \wedge \text{CNF}(p \equiv \neg q) \\
&= (\text{CNF}(p) \otimes \text{CNF}(\neg q)) \wedge (\text{CNF}(\neg p) \otimes \text{CNF}(q)) \wedge \\
&\quad (\text{CNF}(p) \otimes \text{CNF}(\neg\neg q)) \wedge (\text{CNF}(\neg p) \otimes \text{CNF}(\neg q)) \\
&= \{p, \neg q\}, \{\neg p, q\}, \{p, q\}, \{\neg p, \neg q\}
\end{aligned}
$$

Then we apply the DPLL algorithm

$\{\{p,\neg q\},\{\neg p,q\},\{p,q\},\{\neg p,\neg q\}\}$	$\mathcal{I}=\{\}$	No unit clause. Choose literal p (*)
$\{\{p,\neg q\},\{\neg p,q\},\{p,q\},\{\neg p,\neg q\}\}\|_p$	$\mathcal{I}=\{p\}$	
$\{\{q\},\{\neg q\}\}$	$\mathcal{I}=\{p\}$	Apply unit propagation for clause $\{q\}$
$\{\{q\},\{\neg q\}\}\|_q$	$\mathcal{I}=\{p,q\}$	
$\{\{\}\}$	$\mathcal{I}=\{p,q\}$	We have derived the empty clause. So we have backtrack to the last choosen literal (*) and choose the opposite one
$\{\{p,\neg q\},\{\neg p,q\},\{p,q\},\{\neg p,\neg q\}\}$	$\mathcal{I}=\{\}$	Choose literal $\neg p$
$\{\{p,\neg q\},\{\neg p,q\},\{p,q\},\{\neg p,\neg q\}\}\|_{\neg p}$	$\mathcal{I}=\{p\}$	
$\{\{\neg q\},\{q\}\}$	$\mathcal{I}=\{\neg p\}$	
$\{\{q\},\{\neg q\}\}$	$\mathcal{I}=\{\neg p\}$	Apply unit propagation for clause $\{q\}$
$\{\{q\},\{\neg q\}\}\|_q$	$\mathcal{I}=\{\neg p,q\}$	
$\{\{\}\}$	$\mathcal{I}=\{\neg p,q\}$	We have derived the empty clause. Since no backtrack are possible, we stop and return UN-SAT

Since the negation of the formula is unsatisfiable, then the formula must be valid.

□

Exercise 8:
Check the following facts via DPLL

$$\models ((\neg A \vee B) \rightarrow C) \vee ((\neg B \rightarrow A) \rightarrow C)$$

Solution To check that a formula ϕ is valid (i.e., $\models \phi$) with DPLL we have to tranform $\neg\phi$ in CNF and then try to derive the emplty clause. So let's start with transforming

$$\neg(((\neg A \vee B) \rightarrow C) \vee ((\neg B \rightarrow A) \rightarrow C))$$

in CNF obtaining the following clauses

$$\{\neg A, B\}, \{A, B\}, \{\neg C\}$$

By unit propagation we have that $\mathcal{I}(C) = False$, and we have to compute $\{A,B\}|_{\neg C}, \{A,\neg B\}|_{\neg C}$ which remain unchanged. We can therefore choose an assignment $\mathcal{I}(B) = True$ and we obtaine the empty set of clauses. WHich implies that any assignment with $\mathcal{I}$ with $\mathcal{I}(C) = False$ and $\mathcal{I}(B) = True$ falsifies the intitial formula, and therefore the intitial formula is not valid. □

Exercise 9:
Use DPLL to find an assignment that satisfies the clauses of exercise 1.

Solution Let ϕ be the set of clauses of exercise 1.

(1) no unit propagation (there are not unit clauses in ϕ)
(2) select the literal K of the first clasue and assign $\mathcal{I}(K) = True$

(3) compute $\phi|_K$

$$\begin{array}{l} R \vee V \\ \neg R \vee \neg V \\ \neg A \vee R \\ V \\ \neg H \vee A \end{array}$$

(4) apply unit propagation for the clause V setting $\mathcal{I}(V) = True$:

$$\begin{array}{l} \neg R \\ \neg A \vee R \\ \neg H \vee A \end{array}$$

(5) apply unit propagation for the clause $\neg R$ setting $\mathcal{I}(R) = False$:

$$\begin{array}{l} \neg A \\ \neg H \vee A \end{array}$$

(6) apply unit propagation for the clause $\neg A$ setting $\mathcal{I}(A) = False$:

$$\neg H$$

(7) apply unit propagation for the clause $\neg H$ setting $\mathcal{I}(H) = False$:

(8) we obtain the empty set of clauses, this implies DPLL exits with SAT and returns the assignment: $\mathcal{I}(K) = True$, $\mathcal{I}(V) = True$, $\mathcal{I}(R) = False$, $\mathcal{I}(A) = False$, $\mathcal{I}(H) = False$

that corresponds to the situation where K and V are the only one who are chatting. Notice that DPLL tells us that this is a situation compatible with the constraints, but does not guarantee that this is the only one. There could be others. To check if there are other assighment try to start by assigning false to K and continue with the DPLL algorithm. □

Exercise 10:

There are N towns each one having a local radio station. We have to assing a radio frequency out of Q available ones to each radio station. To avoid interferences, towns closer than $20Km$ can not use the same frequency. We have a function $Distance(i, j)$ indicating the distance between town i and j. Is it possible to assign the frequencies? Express the problem in CNF.

Solution We need the set of propositions $\{freq(q, t) \mid t \in T,\ q \in Q\}$, where $freq(q, t)$ means that the frequency q is assigned to the town t.

- Every town need one frequency

$$\bigwedge_{t \in T} \bigvee_{q \in Q} freq(q, t)$$

- the same frequency cannot be assigned to closed towns

$$\bigwedge_{\substack{q, q' \in Q \\ q \neq q'}} \quad \bigwedge_{\substack{t, t' \in T \\ 0 < Distance(t, t') \leq 20Km}} \neg freq(q, t) \vee \neg freq(q, t')$$

- only one frequency is assigned for every town

$$\bigwedge_{t \in T} \bigwedge_{\substack{q, q' \in Q \\ q \neq q'}} \neg freq(q, t) \vee \neg freq(q', t)$$

□

Exercise 11:

Suppose that ϕ has k subformulas $\phi_1, \dots, \phi_k$. How many new propositional variables are introduced by the Tseitin's transformation? **Solution** The Tseitin's

transformation introduces one propositional variable for each non atomic subformula. Therefore, if ϕ contains m propositional variables, the Tseitin's trasformation introduces $n - m$ new propositional variables. □

Exercise 12:

Check if the following formula is valid using DPLL. If it is not valid provide a counter-model, i.e., an assignment that falsify it.

$$(p \wedge q) \vee r \equiv (p \rightarrow \neg q) \rightarrow r$$

Solution To show the validity we have to show that the negated of the formula is not satisfiable. Therefore let us consider the negated of the formula, which is

$$\neg(((p \wedge q) \vee r) \equiv ((p \rightarrow \neg q) \rightarrow r))$$

and transform it in CNF. We use the trnasfromation $\neg(A \equiv B)$ is equivalent ot $(A \vee B) \wedge (\neg A \vee \neg B)$, therefore we obtain the following two formulas that can be transofrmed in CNF independently.

$$(((p \wedge q) \vee r) \vee ((p \rightarrow \neg q) \rightarrow r))$$
$$(\neg((p \wedge q) \vee r) \vee \neg((p \rightarrow \neg q) \rightarrow r)$$

If we tranfrom the first formula we obtain the following clauses

$$\{p, r\}$$
$$\{q, r\}$$

If we transform the second formula we obtain the following clauses

$$\{\neg r\}$$
$$\{\neg p, \neg q\}$$

Therefore the total set of clauses are:

$$\{p, r\}$$
$$\{q, r\}$$
$$\{\neg r\}$$
$$\{\neg p, \neg q\}$$

We can apply unit propagation obtaining

$$\{p\}$$
$$\{q\}$$
$$\{\neg p, \neg q\}$$

and by other two applications of unit propagation we derive the empty clause $\{\}$. Since we have done no branching, then there is no backtrack possible, and DPLL terminates returning UNSAT. Therefore the initial formula was Valid, □

Exercise 13:

Check the following formulas are valid with DPLL.

(1) $(p \to q) \models \neg p \to \neg q$
(2) $(p \to q) \wedge \neg q \models \neg p$
(3) $p \to q \wedge r \models (p \to q) \to r$
(4) $p \vee (\neg q \wedge r) \models q \vee \neg r \to p$
(5) $\neg(p \wedge q) \equiv \neg p \vee \neg q$
(6) $(p \wedge q) \vee r \equiv (p \to \neg q) \to r$
(7) $(p \vee q) \wedge (\neg p \to \neg q) \equiv p$
(8) $((p \to q) \to q) \to q \equiv p \to q$

Exercise 14:

Define a method to transform a propositional formula in a set of formulas of the form

$$\bigwedge_{i=1}^{n} a_i \to \bigvee_{j=1}^{m} b_j \tag{11}$$

where $\bigwedge_{i=1}^{0} a_i = \top$ and $\bigvee_{j=1}^{0} B_j = \bot$.

Solution Notice that, since $A \to B$ is equivalent to $\neg A \vee B$, the formula (11) is equivalent to

$$\neg \bigwedge_{i=1}^{n} a_i \vee \bigvee_{j=1}^{m} b_j$$

by pushing the $\neg$ operator inside the $\bigwedge$, we obtain the following equivalent formula

$$\bigvee_{i=1}^{n} \neg a_i \vee \bigvee_{j=1}^{m} b_j \tag{12}$$

But this form is a clause that contains n negative literals and m positive literals. Therefore to transform a formula ϕ in the form (11) one can first transform ϕ in CNF, and then transform all the clause of the form (12) in the form (11) by applying the inverse transformation that has been shown above. If a cluase does not contain negative literals then it is of the form

$$\bigvee_{i=1}^{m} b_i$$

which is equivalent to

$$\top \to \bigvee_{i=1}^{m} b_i$$

with the convention that $\bigwedge_{i=1}^{0} a_i$ correspond to $\top$ then a clause with no negative literals is transformed in:

$$\bigwedge_{i=1}^{0} a_i \to \bigvee_{i=1}^{m} b_i$$

which is of the form (11). Analogous argument can be done for the clauses that do not contain positive literals. □

Exercise 15:

Provide the Tseitin's Transformation of the following formula:

$$((p \vee q) \rightarrow r) \vee (r \rightarrow (p \vee q))$$

Solution The Tseitin's tranformation introduces one new propositional variable

for all the non atomic subformula of the original formula. The set of non atomic subformulas of $((p \vee q) \rightarrow r) \vee (r \rightarrow (p \vee q))$ (including itself) with the associated new propositional variables are:

$$\begin{array}{lr} x_1 & ((p \vee q) \rightarrow r) \vee (r \rightarrow (p \vee q)) \\ x_2 & ((p \vee q) \rightarrow r) \\ x_3 & (r \rightarrow (p \vee q)) \\ x_4 & (p \vee q) \end{array}$$

We then define the follwong clauses

$$\begin{aligned} & x_1 \equiv x_2 \vee x_3 \\ & x_2 \equiv x_4 \rightarrow r \\ & x_3 \equiv r \rightarrow x_4 \\ & x_4 \equiv p \vee q \end{aligned}$$

and tranform them in clausal form

$$\begin{aligned} & (\neg x_1, x_2, x_3), (\neg x_2, x_1), (\neg x_3, x_1) \\ & (\neg x_2, \neg x_4, r), (x_4, x_2), (\neg r, x_2) \\ & (\neg x_3, \neg r, x_4), (r, x_3), (\neg x_4, x_2) \\ & (\neg x_4, p, q), (\neg p, x_4), (\neg q, x_4) \end{aligned}$$

□

Exercise 16:

Let us consider the formula

$$\phi = ((p \vee q) \wedge r) \rightarrow \neg s \qquad (13)$$

Solution The subformulas of ϕ involved in the tranformation and the corresponding fresh propositional variables are the following:

$$\begin{array}{rl} ((p \wedge q) \vee r) \rightarrow \neg s & x_1 \\ (p \wedge q) \vee r & x_2 \\ p \wedge q & x_3 \end{array}$$

we then apply the sequence of rule (13).

$$\begin{aligned} & x_1 \wedge (x_1 \leftrightarrow ((p \wedge q) \vee r) \rightarrow \neg s) \\ & x_1 \wedge (x_1 \leftrightarrow (x_2 \rightarrow \neg s)) \wedge (x_2 \leftrightarrow ((p \wedge q) \vee r)) \\ & x_1 \wedge (x_1 \leftrightarrow (x_2 \rightarrow \neg s)) \wedge (x_2 \leftrightarrow (x_3 \vee r)) \wedge (x_3 \leftrightarrow (p \wedge q)) \end{aligned}$$

We successifely have to tranform the obtained formula in CNF Obtaining the following set of clauses

$$\begin{aligned}
\mathrm{CNF}(x_1) &= \{x_1\} \\
\mathrm{CNF}(x_1 \leftrightarrow (x_2 \to \neg s)) &= \{\neg x_1, \neg x_2, \neg s\}, \{x_2, x_1\}, \{s, x_1\} \\
\mathrm{CNF}(x_2 \leftrightarrow (x_3 \vee r)) &= \{\neg x_2, x_3, r\}, \{\neg x_3, x_2\}, \{\neg r, x_2\} \\
\mathrm{CNF}(x_3 \leftrightarrow (p \wedge q)) &= \{x_3, p\}, \{x_3, q\}, \{\neg p, \neg q, x_3\}
\end{aligned}$$

□

Bibliography

Davis, Martin, George Logemann, and Donald Loveland (1962). “A machine program for theorem proving”. In: *Communications of the ACM* 5.7, pp. 394–397.

Davis, Martin and Hillary Putnam (1960). “A computing procedure for quantification theory”. In: *Journal of ACM* 7, pp. 201–215.

Tseytin, Grigori (Sept. 1966). *On the complexity of derivation in propositional calculus*. Presented at the Leningrad Seminar on Mathematical Logic. URL: `http://www.decision-procedures.org/handouts/Tseitin70.pdf`.