

Lecture Notes for

Logic and Computability

Course Number: IND04033UF

by

Bettina Könighofer

Bernd Grabner

Belinda Uhl

Contact

Bettina Könighofer

Institute for Applied Information Processing and Communications (IAIK)

Graz University of Technology, Austria

bettina.koenighofer@iaik.tugraz.at

March, 2021



Graz University of Technology

Table of Contents

| | | |
|----------|---|----------|
| 3 | Combinational Equivalence Checking | 1 |
| 3.1 | Translating a Circuit into a Formula | 2 |
| 3.2 | Relations between Satisfiability, Validity, Equivalence and Entailment | 3 |
| 3.3 | Normal Forms | 5 |
| 3.4 | Tseitin Encoding | 7 |
| 3.5 | CEC Example | 9 |
| 3.6 | Exercises | 11 |
| 3.6.1 | Examples | 11 |
| 3.6.2 | Solutions | 12 |
| 3.6.3 | Self-Evaluation | 14 |

3

Combinational Equivalence Checking

In this chapter we discuss combinational equivalence checking (CEC) based on Boolean satisfiability (SAT). CEC plays an important role in the design of electronic systems such as integrated circuits and printed circuit boards. Its immediate application is verifying functional equivalence of combinational circuits after the application of synthesis and optimization tools. In a typical scenario, there are two structurally different implementations of the same design, and the problem is to prove their functional equivalence.

The standard approach that we are discussing in this chapter is reducing the equivalence problem to SAT and using a SAT solver. Though the problem is NP-complete, most verification instances can be solved in reasonable space and time resources due to the high efficiency of modern SAT solvers.

Overview of CEC based on Satisfiability

Let C_1 and C_2 denote the two combinational circuits being checked for equivalence. In order to check whether C_1 and C_2 are equivalent, one has to perform the following steps.

1. The two circuits to be checked for equivalence are encoded into two propositional formulas φ_1 and φ_2 .
2. To check whether φ_1 and φ_2 are equivalent, we use the relation between equivalence and satisfiability and check whether $\varphi_1 \oplus \varphi_2$ is not satisfiable, i.e., $\varphi_1 \equiv \varphi_2$ if and only if $\varphi_1 \oplus \varphi_2$ is UNSAT.
3. SAT solvers operate on formulas given in Conjunctive Normal Form (CNF).

Therefore, in order to use a SAT Solver, the formula $\varphi_1 \oplus \varphi_2$ has to be transformed into a CNF formula by using e.g., Tseitin encoding.

In the remainder of this chapter, we discuss these steps and the needed algorithms and normal forms in detail.

3.1 Translating a Circuit into a Formula

Each gate of the circuit can be represented via a propositional formula over its inputs and output. The following figure lists a summary of the common Boolean logic gates with their symbols and truth tables.

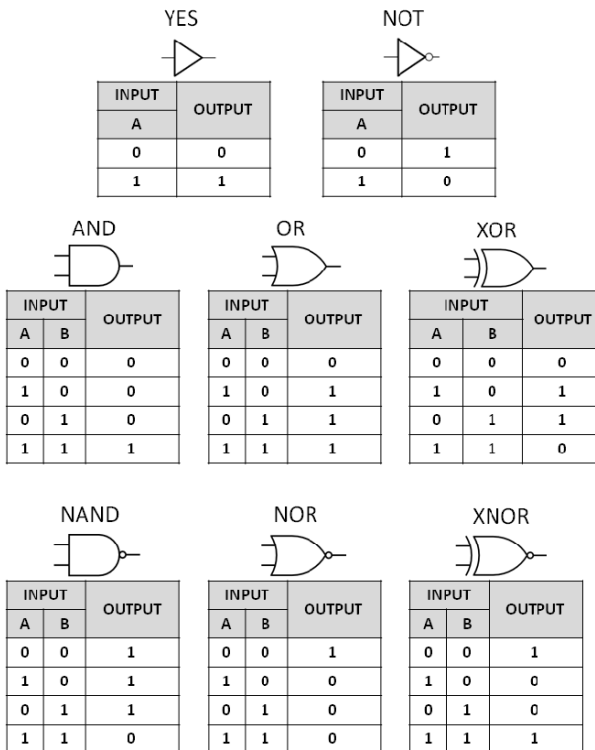
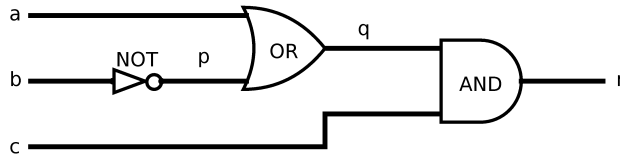


Figure 3.1: Boolean logic gates.

Example. Translate the following circuit C into a propositional formula.



The inputs of C are denoted by a , b , and c and the output is denoted by r . We assign temporary variable names to the inner wires; in this case we use p and q . Using these variables, we can create the propositional formula over the inputs and the output of C .

$$\begin{aligned} r &= q \wedge c \\ &= (a \vee p) \wedge c \\ &= (a \vee \neg b) \wedge c \end{aligned} \tag{3.1}$$

3.2 Relations between Satisfiability, Validity, Equivalence and Entailment

In the first chapter, we discussed the notions of satisfiability, validity, equivalence, and semantic entailment. *All of these notions can be reduced to each other. Therefore, only one decision procedure is needed to decide all notions.* This is in particular important, since for the question of satisfiability there exist tools (SAT solvers) that are able to decide most practical instances highly efficiently, even though the problem of deciding satisfiability is NP-complete. Therefore, in order to answer the questions of validity, semantic entailment, or equivalence, these questions will most of the time be reduced to the question of satisfiability.

In this section, we discuss the relations between all notions.

Duality of Satisfiability and Validity

A formula φ is valid, if and only if, $\neg\varphi$ is not satisfiable.

Consider the formula $\varphi = (x \vee \neg x)$. This formula is valid, i.e., all rows in the truth table would evaluate to *true*. The negation of φ is the following: $\neg\varphi = \neg(x \vee \neg x) = \neg x \wedge x$, which is not satisfiable, i.e., all rows in the truth table would evaluate to *false*.

A formula φ is satisfiable, if and only if, $\neg\varphi$ is not valid.

If φ is satisfiable, there is at least one model that makes the formula true. If we negate the formula, these models make the negated formula false, and therefore, the negated formula cannot be valid.

| | | |
|------------------|-----------------------------|-----------------------------------|
| solving using | φ satisfiable? | φ valid? |
| Satisfiability | ✓ | $\neg\varphi$ not satisfiable? |
| Validity | $\neg\varphi$ not valid? | ✓ |

Overview over all Relations

| solving using | φ satisfiable? | φ valid? | $\varphi \models \psi$? | $\varphi \equiv \psi$? |
|------------------|----------------------------------|-----------------------------------|---|--|
| Satisfiability | ✓ | $\neg\varphi$ not satisfiable? | $\varphi \wedge \neg\psi$ not satisfiable? | $\varphi \oplus \psi$ not satisfiable? |
| Validity | $\neg\varphi$ not valid? | ✓ | $\varphi \rightarrow \psi$ valid? | $\varphi \leftrightarrow \psi$ valid? |
| Entailment | $\top \not\models \neg\varphi$? | $\top \models \varphi$? | ✓ | $\varphi \models \psi$ and $\psi \models \varphi$? |
| Equivalence | $\varphi \not\equiv \perp$? | $\varphi \equiv \top$? | $\varphi \rightarrow \psi \equiv$ \top ? | ✓ |

Row 1: Decide entailment using satisfiability. The question whether $\varphi \models \psi$ can be decided by checking $\varphi \wedge \neg\psi$ not satisfiable. If there does not exist a counterexample, i.e., a model that makes φ true and makes ψ false, then we have semantic entailment.

Row 1: Decide equivalence using satisfiability. The question whether $\varphi \equiv \psi$ can be decided by checking $\varphi \oplus \psi$ not satisfiable. If there does not exist a model, that makes one sub-formula true, and the other sub-formula false, then the two formulas are semantically equivalent.

Row 2: Decide entailment using validity. The question whether $\varphi \models \psi$ can be decided by checking $\varphi \rightarrow \psi$ is valid. For an implication to hold, all models that satisfy φ also have to satisfy ψ . This is also the requirement for $\varphi \models \psi$ to be true.

Row 2: Decide equivalence using validity. The question whether $\varphi \equiv \psi$ can be decided by checking $\varphi \leftrightarrow \psi$ is valid. This holds, since the formula $\varphi \leftrightarrow \psi$ only evaluates to true under models that either make both sub-formulas true, or both false.

Row 3: Decide satisfiability using entailment. The question whether φ is satisfiable can be decided by checking $\top \not\models \neg\varphi$. If not all models satisfy $\neg\varphi$, then there exists a model that satisfies φ , and therefore φ must be satisfiable.

Row 3: Decide validity using entailment. The question whether φ is valid can be decided by checking $\top \models \varphi$. If all models satisfy φ , then φ must be valid.

Row 3: Decide equivalence using entailment. The question whether $\varphi \equiv \psi$ is valid can be decided by checking $\varphi \models \psi$ and $\psi \models \varphi$. If all models that

satisfy φ also satisfy ψ and all models that satisfy ψ also satisfy φ , then φ and ψ are satisfied by exactly the same models and are equivalent.

Row 4: Decide satisfiability using equivalence. The question φ is satisfiable can be decided by checking $\varphi \not\equiv \perp$. If it is not true, that φ evaluates to false under all models, then φ is satisfiable.

Row 4: Decide validity using equivalence. The question whether φ is valid can be decided by checking $\varphi \equiv \top$. If it is true, that φ evaluates to true under all models and is valid.

Row 4: Decide entailment with equivalence. The question whether $\varphi \models \psi$ can be decided by checking $\varphi \rightarrow \psi \equiv \top$. If all models that satisfy φ also satisfy ψ , then we have semantic entailment.

3.3 Normal Forms

For ease of implementation, most SAT solvers operate on formulas given in conjunctive normal form (CNF). In this section we discuss the two standard normal forms: the disjunctive normal form (DNF) and the conjunctive normal form (CNF).

Terminology

Let φ be a propositional formula defined over Boolean variables x_1, \dots, x_n . We introduce the following symbols and terminology to define CNF and DNF.

- A *literal* is one of the variables x_i or the negation of a variable.
- A literal is called *positive* if it is just a variable. A literal is called *negative* if it is the negation of a variable.
- A *clause* is a disjunction of literals, e.g., $x_1 \vee x_2$.
- A *cube* is a conjunction of literals, e.g., $x_1 \wedge x_2$.

Disjunctive Normal Form (DNF)

A formula in DNF is a *disjunction of cubes*, e.g., $(p \wedge q \wedge r) \vee (\neg p \wedge q \wedge r) \vee (\neg q)$.

Any formula in propositional logic can be converted to the disjunctive normal form. One simple way to do so is to use the truth table of the formula. For every model/line that delivers a 1, a *conjunction* of all literals of that line is formed, i.e., variables which are assigned 1 in the line are not negated and variables which are assigned 0 are negated. These conjunctions per line form the cubes of the DNF formula. By connecting all cubes with conjunctions, ones obtains the final DNF formula.

Example. Given the formula $\varphi := \neg a \vee \neg(b \rightarrow c)$. Compute its representation in DNF using its truth table.

| a | b | c | $\neg a \vee \neg(b \rightarrow c)$ |
|-----|-----|-----|-------------------------------------|
| F | F | F | T |
| F | F | T | T |
| F | T | F | T |
| F | T | T | T |
| T | F | F | F |
| T | F | T | F |
| T | T | F | T |
| T | T | T | F |

The resulting formula in DNF is

$$(\neg a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c).$$

Conjunctive Normal Form (CNF)

A formula in CNF is a *conjunction of clauses*, e.g., $(p \vee q \vee r) \wedge (\neg p \vee q \vee r)$.

To convert the formula to CNF, we consider the models/lines that make the formula *false*. For every line that delivers a 0, a *disjunction* of the *negated literals* of that lines is formed, i.e., variables which are assigned 1 in the line are negated and variables which are assigned 0 are not negated. These disjunctions per line form the clauses of the CNF formula. By connecting all clauses with disjunctions, ones obtains the final CNF formula.

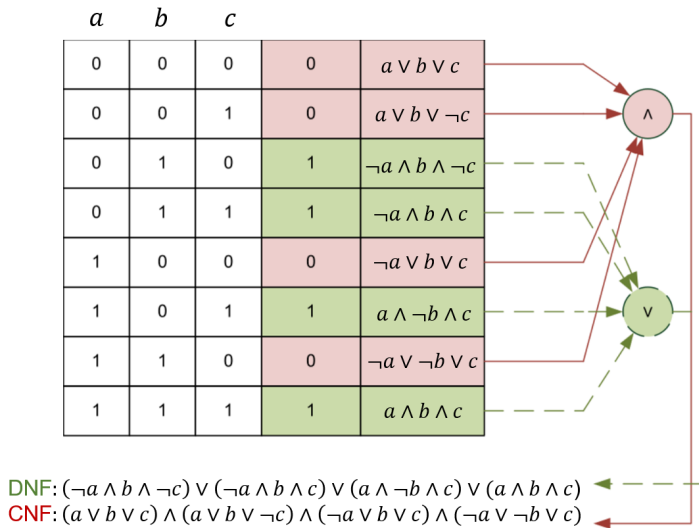
Example. Given the formula $\varphi := \neg a \vee \neg(b \rightarrow c)$. Compute its representation in CNF using its truth table.

| a | b | c | $\neg a \vee \neg(b \rightarrow c)$ |
|-----|-----|-----|-------------------------------------|
| F | F | F | T |
| F | F | T | T |
| F | T | F | T |
| F | T | T | T |
| T | F | F | F |
| T | F | T | F |
| T | T | F | T |
| T | T | T | F |

The resulting formula in DNF is

$$(\neg a \vee b \vee c) \wedge (\neg a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee \neg c).$$

Example. Provide the formula in CNF and in DNF that is represented by the following truth table.

Figure 3.2: DNF/CNF example.¹

3.4 Tseitin Encoding

Propositional formulas need to be converted into CNF before they can be given to a SAT solver. It is always possible to transform a Boolean formula into CNF that is equivalent by using for example the truth table to convert the formula. However, there are propositional formulas whose equivalent representation in CNF is necessarily *exponential*. Therefore, we do not generate a CNF that is equivalent, but instead perform a transformation that *only preserves satisfiability*. We call such a formula that is not equivalent to the original formula but preserves satisfiability *equisatisfiable*.

Definition. Two propositional formulas φ and ψ are *equisatisfiable* if and only if either *both are satisfiable* or *both are unsatisfiable*.

One way to perform the conversion of a propositional formula into an equisatisfiable formula in CNF is the *Tseitin's method*, which only results in a *linear* blow-up.

The Tseitin Algorithm

Tseitin's method takes the syntax tree for a propositional formula φ as input. An internal node in this tree is a Boolean connective while a leaf is a Boolean variable. The algorithm traverses the tree, beginning with the leaves, and associates a *new variable* to each node (i.e., to each subformula). By traversing through the tree, the algorithm collects a set of clauses.

¹Taken from https://de.wikipedia.org/wiki/Konjunktive_Normalform#/media/Datei:Knf+dnf.svg by JensKhol, licensed under CC-BY-SA-2.0

We will explain the Tseitin's method for formulas that are restricted to the Boolean connectives \wedge , \vee , and \neg .

Step 1: Assigning new variables to all nodes in the parse tree / to each subformula.

Example. Consider the formula $\varphi := ((p \vee q) \wedge r) \vee \neg p$. For each sub-formula, we introduce a fresh variable.

$$\begin{array}{c} ((\underbrace{p \vee q}_{x_1}) \wedge r) \vee \underbrace{\neg p}_{x_3} \\ \underbrace{\hspace{1.5cm}}_{x_2} \\ \underbrace{\hspace{2.5cm}}_{x_\varphi} \end{array}$$

Step 2: Add new clauses for each new variable.

Each new variable represents either a \neg , \wedge , or \vee node in the parse tree / subformula of the form $\neg\varphi$, $\varphi \wedge \psi$, or $\varphi \vee \psi$. The *Tseitin-Rewriting Rules* define, which clauses are associated with which type of node / subformula.

Tseitin-Rewrite rules:

For each node, let x be the new variable, and let p and q be the new variables for the two subformulas of the node.

- \wedge node: for $x \leftrightarrow (p \wedge q)$ introduce the clauses $(\neg x \vee p) \wedge (\neg x \wedge q) \wedge (\neg p \vee \neg q \vee x)$.
- \vee node: for $x \leftrightarrow (p \vee q)$ introduce the clauses $(\neg p \vee x) \wedge (\neg q \vee x) \wedge (\neg x \vee p \vee q)$.
- \neg node: for $x \leftrightarrow \neg p$ introduce the clauses $(\neg x \vee \neg p) \wedge (p \vee x)$.

Finally, to obtain the final equisatisfiable formula in CNF, we connect the variable for the top level node of the formula and all generated clauses with conjunctions.

Using the Tseitin algorithm and its rewriting rules, we would obtain the following clauses for the example above with $\varphi := ((p \vee q) \wedge r) \vee \neg p$:

$$\begin{aligned} CNF(\varphi) = & (\neg p \vee x_1) \wedge (\neg q \vee x_1) \wedge (\neg x_1 \vee p \vee q) \\ & \wedge (\neg x_2 \vee x_1) \wedge (\neg x_2 \wedge r) \wedge (\neg x_1 \vee \neg r \vee x_2) \\ & \wedge (\neg x_3 \vee \neg p) \wedge (p \vee x_3) \\ & \wedge (\neg x_2 \vee x_\varphi) \wedge (\neg x_3 \vee x_\varphi) \wedge (\neg x_\varphi \vee x_2 \vee x_3) \\ & \wedge x_\varphi \end{aligned} \tag{3.2}$$

Derivation of the Tseitin Rewriting Rules

Following we briefly justify the clauses that are generated for \wedge , \vee , and \neg nodes.

- $x \leftrightarrow (p \wedge q)$ generates the clauses $(\neg x \vee p) \wedge (\neg x \vee q) \wedge (\neg p \vee \neg q \vee x)$ because:

$$\begin{array}{l|l}
x \leftrightarrow (p \wedge q) & \\
(x \rightarrow (p \wedge q)) \wedge ((p \wedge q) \rightarrow x) & | \phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi) \\
(x \rightarrow p) \wedge (x \rightarrow q) \wedge ((p \wedge q) \rightarrow x) & | \phi \rightarrow (\psi \wedge \chi) \equiv (\phi \rightarrow \psi) \wedge (\phi \rightarrow \chi) \\
(\neg x \vee p) \wedge (\neg x \wedge q) \wedge (\neg(p \wedge q) \vee x) & | \phi \rightarrow \psi \equiv \neg\phi \vee \psi \\
(\neg x \vee p) \wedge (\neg x \wedge q) \wedge (\neg p \vee \neg q \vee x) & | \neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi
\end{array}$$

- $x \leftrightarrow (p \vee q)$ generates the clauses $(\neg p \vee x) \wedge (\neg q \vee x) \wedge (\neg x \vee p \vee q)$ because:

$$\begin{array}{l|l}
x \leftrightarrow (p \vee q) & \\
(x \rightarrow (p \vee q)) \wedge ((p \vee q) \rightarrow x) & | \phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi) \\
(x \rightarrow (p \vee q)) \wedge (p \rightarrow x) \wedge (q \rightarrow x) & | \psi \vee \phi \rightarrow \chi \equiv (\phi \rightarrow \chi) \wedge (\psi \rightarrow \chi) \\
(\neg x \vee p \vee q) \wedge (\neg p \vee x) \wedge (\neg q \vee x) & | \phi \rightarrow \psi \equiv \neg\phi \vee \psi \\
(\neg p \vee x) \wedge (\neg q \vee x) \wedge (\neg x \vee p \vee q) & | \text{rearranging}
\end{array}$$

- $x \leftrightarrow \neg p$ generates the clauses $(\neg x \vee \neg p) \wedge (p \vee x)$ because:

$$\begin{array}{l|l}
x \leftrightarrow \neg p & \\
(x \rightarrow \neg p) \wedge (\neg p \rightarrow x) & | \phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi) \\
(\neg x \vee \neg p) \wedge (\neg \neg p \vee x) & | \phi \rightarrow \psi \equiv \neg\phi \vee \psi \\
(\neg x \vee \neg p) \wedge (p \vee x) & | \neg\neg\phi \equiv \phi
\end{array}$$

Summary: Tseitin Rewriting Rules

$$\begin{array}{lcl}
\chi \leftrightarrow (\varphi \vee \psi) & \Leftrightarrow & (\neg\varphi \vee \chi) \wedge (\neg\psi \vee \chi) \wedge (\neg\chi \vee \varphi \vee \psi) \\
\chi \leftrightarrow (\varphi \wedge \psi) & \Leftrightarrow & (\neg\chi \vee \varphi) \wedge (\neg\chi \vee \psi) \wedge (\neg\varphi \vee \neg\psi \vee \chi) \\
\chi \leftrightarrow \neg\varphi & \Leftrightarrow & (\neg\chi \vee \neg\varphi) \wedge (\varphi \vee \chi)
\end{array}$$

Properties of Tseitin Encoding

We make two observations about the set of clauses that is generated by the Tseitin procedure for a formula φ .

1. The number of variables and clauses is *linear* in the size of φ . We have thus avoided the exponential blowup we would have observed when constructing equivalent CNF formula.
2. The constructed formula $CNF(\varphi)$ is equisatisfiable to φ : $CNF(\varphi)$ has a satisfying assignment if and only if there is a satisfying assignment for φ . We can obtain a satisfying assignment for φ from the satisfying assignment for $CNF(\varphi)$ by simply dropping the additional variables that the algorithm has introduced.

3.5 CEC Example

Example. Given are two formulas $\varphi_1 = \neg a \wedge \neg b$ and $\varphi_2 = \neg(a \vee b)$. Check whether φ_1 and φ_2 are semantically equivalent using the reduction to satisfiability.

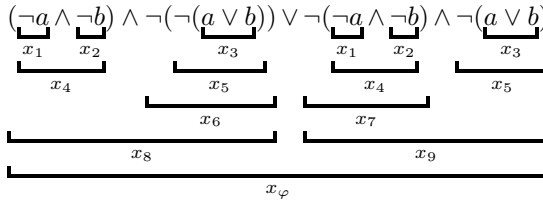
We need to check, whether $\varphi_1 \oplus \varphi_2$ is not satisfiable, i.e. $\varphi_1 \equiv \varphi_2$ if and only if $\varphi_1 \oplus \varphi_2$ is UNSAT. Therefore, we perform the following steps.

- We construct the formula φ :

$$\begin{aligned} \varphi &= \varphi_1 \oplus \varphi_2 = (\neg a \wedge \neg b) \oplus \neg(a \vee b) \\ &= (\neg a \wedge \neg b) \wedge \neg(\neg(a \vee b)) \vee \neg(\neg a \wedge \neg b) \wedge \neg(a \vee b) \end{aligned} \quad (3.3)$$

- Next, the formula φ has to be transformed into a CNF formula by using Tseitin encoding.

Tseitin-Step 1: Assigning new variables to subformulas.



Tseitin-Step 2: Apply rewriting rules.

$$CNF(\varphi) = x_\varphi \wedge$$

$$\begin{aligned} &(\neg x_8 \vee x_\varphi) \wedge (\neg x_9 \vee x_\varphi) \wedge (\neg x_\varphi \vee x_9 \vee x_8) \wedge & | \text{rewriting } x_\varphi \\ &(\neg x_9 \vee x_7) \wedge (\neg x_9 \vee x_5) \wedge (\neg x_7 \vee \neg x_5 \vee x_9) \wedge & | \text{rewriting } x_9 \\ &(\neg x_8 \vee x_4) \wedge (\neg x_8 \vee x_6) \wedge (\neg x_4 \vee \neg x_6 \vee x_8) \wedge & | \text{rewriting } x_8 \\ &(\neg x_7 \vee \neg x_4) \wedge (x_7 \vee x_4) \wedge & | \text{rewriting } x_7 \\ &(\neg x_6 \vee \neg x_5) \wedge (x_6 \vee x_5) \wedge & | \text{rewriting } x_6 \\ &(\neg x_5 \vee \neg x_3) \wedge (x_5 \vee x_3) \wedge & | \text{rewriting } x_5 \\ &(\neg x_4 \vee x_1) \wedge (\neg x_4 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge & | \text{rewriting } x_4 \\ &(\neg a \vee x_3) \wedge (\neg b \vee x_3) \wedge (\neg x_3 \vee a \vee b) \wedge & | \text{rewriting } x_3 \\ &(\neg x_2 \vee \neg b) \wedge (x_2 \vee b) \wedge & | \text{rewriting } x_2 \\ &(\neg x_1 \vee \neg a) \wedge (x_1 \vee a) & | \text{rewriting } x_1 \end{aligned}$$

- Finally, the formula $CNF(\varphi)$ is given to a SAT solver. If the SAT solver determines that $CNF(\varphi)$ is unsatisfiable, then $\varphi_1 \equiv \varphi_2$. If the SAT solver determines that $CNF(\varphi)$ is satisfiable, then $\varphi_1 \not\equiv \varphi_2$.

3.6 Exercises

3.6.1 Examples

Example 1. Construct the DNF and the CNF from the following truth table:

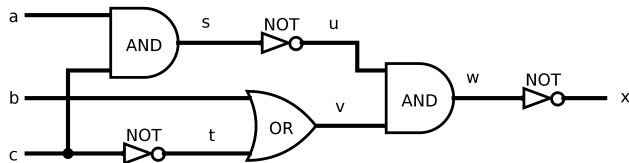
| a | b | c | φ |
|-----|-----|-----|-----------|
| F | F | F | F |
| F | F | T | F |
| F | T | F | T |
| F | T | T | F |
| T | F | F | T |
| T | F | T | T |
| T | T | F | F |
| T | T | T | T |

Example 2. Transform the following formulas into equisatisfiable formulas in CNF using Tseitin encoding.

(a) $\varphi = \neg(p \wedge (q \vee r))$

(b) $\varphi = (p \vee q) \wedge \neg(\neg p \wedge \neg r)$

Example 3. Compute the propositional formula for the following circuit and transform it into an equisatisfiable formulas in CNF using Tseitin encoding.



Example 4. Given are two formulas $\varphi_1 = a \wedge b \vee \neg a \wedge \neg b$ and $\varphi_2 = a \vee b \rightarrow a \wedge b$. Check whether φ_1 and φ_2 are semantically equivalent using the reduction to satisfiability.

3.6.2 Solutions

Solution 1. $DNF(\varphi) = (\neg a \wedge b \wedge c) \vee (a \wedge \neg b \wedge \neg c) \vee (a \wedge \neg b \wedge c) \vee (a \wedge b \wedge c)$
 $CNF(\varphi) = (a \vee b \vee c) \wedge (a \vee b \vee \neg c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee \neg b \vee \neg c)$

Solution 2. (a) $\neg(p \wedge (q \vee r))$

$$CNF(\varphi) = x_\varphi \wedge (\neg x_\varphi \vee \neg x_2) \wedge (x_\varphi \vee x_2) \wedge (\neg x_2 \vee p) \wedge (\neg x_2 \vee x_1) \wedge (\neg p \vee \neg x_1 \vee x_2) \wedge (\neg q \vee x_1) \wedge (\neg r \vee x_1) \wedge (\neg x_1 \vee q \vee r)$$

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

$$CNF(\varphi) = x_\varphi \wedge (\neg x_\varphi \vee x_5) \wedge (\neg x_\varphi \vee x_4) \wedge (\neg x_5 \vee \neg x_4 \vee x_\varphi) \wedge (\neg p \vee x_5) \wedge (\neg q \vee x_5) \wedge (\neg x_5 \vee p \vee q) \wedge (\neg x_4 \vee \neg x_3) \wedge (x_4 \vee x_3) \wedge (\neg x_3 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg r) \wedge (x_2 \vee r) \wedge (\neg x_1 \vee \neg p) \wedge (x_1 \vee p)$$

Solution 3. $\neg(\neg(a \wedge c) \wedge (b \vee \neg c))$

$$CNF(\varphi) = x_\varphi \wedge (\neg x_\varphi \vee \neg x_5) \wedge (x_\varphi \vee x_5) \wedge (\neg x_5 \vee x_4) \wedge (\neg x_5 \vee x_2) \wedge (\neg x_4 \vee \neg x_2 \vee x_5) \wedge (\neg x_4 \vee \neg x_3) \wedge (x_4 \vee x_3) \wedge (\neg x_3 \vee a) \wedge (\neg x_3 \vee c) \wedge (\neg a \vee \neg c \vee x_3) \wedge (\neg b \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee b \vee \neg x_1) \wedge (\neg x_1 \vee \neg c) \wedge (x_1 \vee c)$$

Solution 4. Transformation of the input formula:

$$\begin{aligned}
 & a \wedge b \vee \neg a \wedge \neg b a \vee b \rightarrow a \wedge b \\
 & a \wedge b \vee \neg a \wedge \neg b \neg(a \vee b) \vee a \wedge b \\
 & (a \wedge b \vee \neg a \wedge \neg b) \wedge \neg(\neg(a \vee b) \vee a \wedge b) \vee \neg(a \wedge b \vee \neg a \wedge \neg b) \wedge \\
 & \quad (\neg(a \vee b) \vee a \wedge b)
 \end{aligned}$$

Tseitin Step 1:

$$\begin{array}{c}
 \underbrace{\underbrace{\underbrace{a \wedge b}_{x_1} \vee \underbrace{\underbrace{\neg a \wedge \neg b}_{x_2, x_3}}_{x_5}}_{x_7} \wedge \neg(\underbrace{\underbrace{\neg(a \vee b)}_{x_4} \vee \underbrace{a \wedge b}_{x_1}}_{x_6}) \vee \neg(\underbrace{\underbrace{a \wedge b}_{x_1} \vee \underbrace{\underbrace{\neg a \wedge \neg b}_{x_2, x_3}}_{x_5}}_{x_7}) \wedge \underbrace{\underbrace{\neg(a \vee b)}_{x_4} \vee \underbrace{a \wedge b}_{x_1}}_{x_6}}_{x_9} \\
 \underbrace{\underbrace{\underbrace{\underbrace{\neg(a \vee b)}_{x_4} \vee \underbrace{a \wedge b}_{x_1}}_{x_6}}_{x_8} \vee \underbrace{\underbrace{\underbrace{a \wedge b}_{x_1} \vee \underbrace{\underbrace{\neg a \wedge \neg b}_{x_2, x_3}}_{x_5}}_{x_7}}_{x_10}}_{x_11} \wedge \underbrace{\underbrace{\underbrace{\underbrace{\neg(a \vee b)}_{x_4} \vee \underbrace{a \wedge b}_{x_1}}_{x_6}}_{x_8}}_{x_12}}_{x_\varphi}
 \end{array}$$

Tseitin Step 2:

$$\begin{aligned}
 CNF(\varphi) = x_\varphi \wedge & \\
 (\neg x_{11} \vee x_\varphi) \wedge (\neg x_{12} \vee x_\varphi) \wedge (\neg x_\varphi \vee x_{11} \vee x_{12}) \wedge & |x_\varphi \\
 (\neg x_{12} \vee x_{10}) \wedge (\neg x_{12} \vee x_8) \wedge (\neg x_{10} \vee \neg x_8 \vee x_{12}) \wedge & |x_{12} \\
 (\neg x_{11} \vee x_7) \wedge (\neg x_{11} \vee x_9) \wedge (\neg x_7 \vee \neg x_9 \vee x_{11}) \wedge & |x_{11} \\
 (\neg x_7 \vee \neg x_{10}) \wedge (x_7 \vee x_{10}) \wedge & |x_{10} \\
 (\neg x_8 \vee \neg x_9) \wedge (x_8 \vee x_9) \wedge & |x_9 \\
 (\neg x_6 \vee x_8) \wedge (\neg x_1 \vee x_8) \wedge (\neg x_8 \vee x_6 \vee x_1) \wedge & |x_8 \\
 (\neg x_1 \vee x_7) \wedge (\neg x_5 \vee x_7) \wedge (\neg x_7 \vee x_1 \vee x_5) \wedge & |x_7 \\
 (\neg x_4 \vee \neg x_6) \wedge (x_4 \vee x_6) \wedge & |x_6 \\
 (\neg x_5 \vee x_2) \wedge (\neg x_5 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_5) \wedge & |x_5 \\
 (\neg a \vee x_4) \wedge (\neg b \vee x_4) \wedge (\neg x_4 \vee a \vee b) \wedge & |x_4 \\
 (\neg b \vee \neg x_3) \wedge (b \vee x_3) \wedge & |x_3 \\
 (\neg a \vee \neg x_2) \wedge (a \vee x_2) \wedge & |x_2 \\
 (\neg x_1 \vee a) \wedge (\neg x_1 \vee b) \wedge (\neg a \vee \neg b \vee x_1) & |x_1
 \end{aligned}$$

3.6.3 Self-Evaluation

Example 1. SAT solvers usually require input formulas to be in *Conjunctive Normal Form* (CNF). In the following list, tick all items that conform to CNF.

(Remark: Lower-case Roman letters represent propositional variables.)

- $(a \wedge b \wedge \neg c) \vee (\neg b \wedge \neg c) \vee (e \wedge \neg f)$
- a
- $\neg b$
- $a \wedge \neg b$
- $a \vee \neg b$
- $a \vee (\neg b \wedge c)$
- $(a \vee \neg b) \wedge c$
- $\neg(p \vee q)$
- $x \vee \neg y \vee z$
- A formula φ that consists of a conjunction of clauses c_1, c_2, \dots, c_n .
- A formula ψ that consists of a disjunction of clauses c_1, c_2, \dots, c_n .
- A formula χ that consists of a conjunction of cubes c_1, c_2, \dots, c_n .
(Hint: A “cube” is a conjunction of literals.)
- A formula ξ that consists of a disjunction of cubes c_1, c_2, \dots, c_n .
- A literal l .

Example 2. Tick each correct ending of the following sentence. “A *Conjunctive Normal Form* (CNF) is ...

- ... a conjunction of disjunctions of literals.”
- ... a conjunction of clauses.”
- ... a formula that consists only of logical AND operations on sub-formulas which only consist of OR operations on just variables and negations of variables.”

Example 3. Consider the propositional formula $\varphi = (\neg(\neg a \wedge b) \vee \neg c)$. Fill out the truth table for φ (and its subformulas). Write **T** for true and **F** for false. Compute a CNF for φ from the truth table. Explain the steps that you need to perform.

| a | b | c | $\neg a$ | $\neg a \wedge b$ | $\neg(\neg a \wedge b)$ | $\neg c$ | $\varphi = (\neg(\neg a \wedge b) \vee \neg c)$ |
|----------|----------|----------|----------|-------------------|-------------------------|----------|---|
| F | F | F | | | | | |
| F | F | T | | | | | |
| F | T | F | | | | | |
| F | T | T | | | | | |
| T | F | F | | | | | |
| T | F | T | | | | | |
| T | T | F | | | | | |
| T | T | T | | | | | |

Example 4. Why are truth tables, in general, not used to compute CNFs of formulas?

Example 5. Consider two combinational circuits. Explain an **efficient way** how to check whether the two circuits are equivalent, using a propositional SAT solver (i.e. without using a truth table)

Example 6. (a) Suppose you have a propositional formula φ . Let ψ be the result of applying Tseitin's encoding to φ . Is φ *equivalent* to ψ ?

Yes

No

(b) Provide a reason for your answer.

Example 7. What is the big advantage of applying Tseitin encoding to obtain a CNF?

Example 8. Derive a Rewrite-Rule for an implication node, i.e., what clauses are introduced by the node $x \leftrightarrow (p \rightarrow q)$?

Example 9. Apply Tseitin's encoding to the following formula:

$$\Phi := \neg \left[(a \vee \neg b) \wedge \neg ((a \vee \neg c) \vee d) \right]$$