Lecture Notes for

Logic and Computability

Course Number: IND04033UF

Contact

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

Graz University of Technology

5 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 for checking the equivalence of two circuits is to reduce the equivalence problem to SAT and using a solver to decide the problem instance.

Overview of CEC based on Satisfiablilty

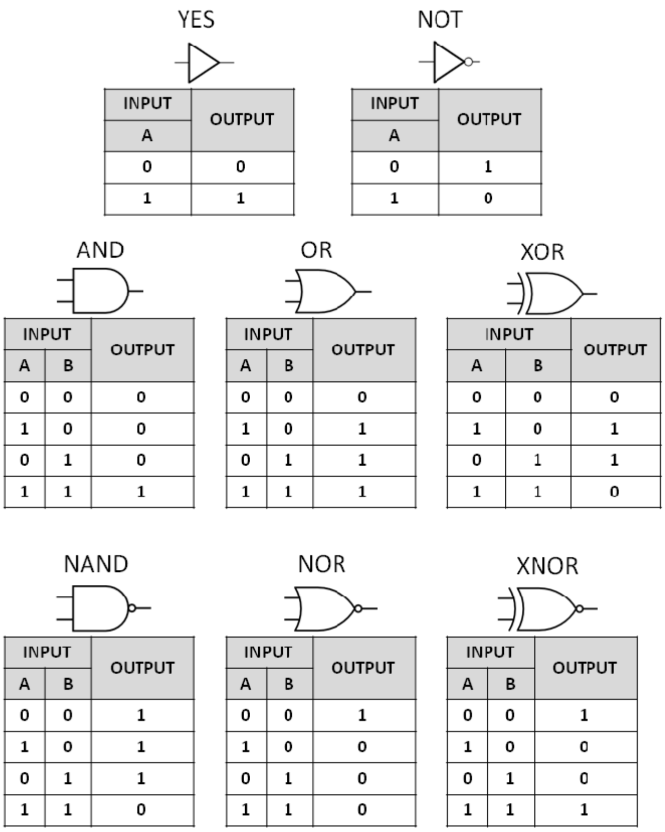
In the following, we give the overview of the algorithm to check for the equivalence of two circuits. In the remainder of this chapter, we will discuss the individual steps in more detail.

Algorithm - Decide equivalence of combinational circuits. Let C_1 and C_2 denote the two combinational circuits. In order to check whether C_1 and C_2 are equivalent, we perform the following steps:

1. Translate C_1 and C_2 into propositional formulas φ_1 and φ_2.
2. Compute the CNF of $\varphi_1 \oplus \varphi_2$, using *Tseitin encoding*; i.e., $\mathrm{CNF}(\varphi_1 \oplus \varphi_2)$.
3. Use $\mathrm{CNF}(\varphi_1 \oplus \varphi_2)$ as input for a SAT solver and determine satisfiability.
4. C_1 and C_2 are equivalent if and only if $\mathrm{CNF}(\varphi_1 \oplus \varphi_2)$ is UNSAT.

5.1 Translation of Circuits to Formulas

Each gate of a combinational circuit can be represented via a propositional formula over its inputs and outputs. The following figure gives a summary of common Boolean logic gates with their representation in a curcuit and the corresponding truth tables.



YES

INPUT	OUTPUT
A	
0	0
1	1

NOT

INPUT	OUTPUT
A	
0	1
1	0

AND

INPUT		OUTPUT
A	B	
0	0	0
1	0	0
0	1	0
1	1	1

OR

INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

XOR

INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

NAND

INPUT		OUTPUT
A	B	
0	0	1
1	0	1
0	1	1
1	1	0

NOR

INPUT		OUTPUT
A	B	
0	0	1
1	0	0
0	1	0
1	1	0

XNOR

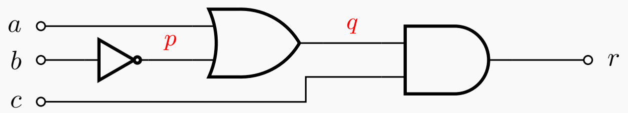
INPUT		OUTPUT
A	B	
0	0	1
1	0	0
0	1	0
1	1	1

Figure 5.1: Boolean logic gates.

When translating a curcuit into a propositional formula, we want to find the formula that expresses the output depending on the inputs. In order to translate a curcuit, we introduce temporary variables as intermediate results of gates. The following example shows the individual steps.

Example 1

Translate the following circuit C into a propositional formula.



Solution. 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.

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

5.2 Relations between Satisfiability, Validity, Equivalence and Semantic Entailment

In the first chapter, we have introduced the notions of satisfiability, validity, equivalence, and semantic entailment. *Each of these notions can be reduced to every other. Therefore, only one decision procedure is needed to decide all notions.* This is particularly important, since deciding the satisfiability of a formula can be done efficiently with SAT solvers. Therefore, in order to answer whether a formula is valid, whether two formulas are equivalent or whether a formula semanticly entails another formula can be decided by checking satisfiability.

In this section, we discuss the relations between the 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 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.

Overview over all Relations

using \ solving	φ satisfiable?	φ valid?	$\varphi \models \psi$?	$\varphi \equiv \psi$?
Satisfiability	✓	$\neg\varphi$ not satisfiable?	$\varphi \land \neg\psi$ 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	$\varphi \not\equiv \bot$?	$\varphi \equiv \top$?	$\varphi \rightarrow \psi \equiv \top$?	✓

Deciding semantic entailment via satisfiability. The question whether $\varphi \models \psi$ can be decided by checking whether $\varphi \lor \neg\psi$ not satisfiable. If there is no model under which φ evaluates true and ψ evaluates to false, φ semantically entails ψ.

Deciding equivalence via satisfiability. The question whether $\varphi \equiv \psi$ can be decided by checking whether $\varphi \oplus \psi$ is not satisfiable. If there is no model, under which one sub-formula evaluates to true and the other sub-formula evaluates to false, then the two formulas are semantically equivalent. *We are going to decide whether two curcuits are equivalent using this insight.*

Deciding semantic entailment via validity. The question whether $\varphi \models \psi$ can be decided by checking whether $\varphi \rightarrow \psi$ 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.

Deciding equivalence via validity. The question whether $\varphi \equiv \psi$ can be decided by checking whether $\varphi \leftrightarrow \psi$ 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.

Deciding satisfiability via semantic entailment. The question whether φ is satisfiable can be decided by checking $\mathbf{T} \not\models \neg\varphi$. If not all models satisfy $\neg\varphi$, then there exists a model that satisfies φ, and therefore φ is satisfiable.

Deciding validity via semantic entailment. The question whether φ is valid can be decided by checking $\top \models \varphi$. If all models satisfy φ, then φ is valid.

Deciding equivalence via 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 therefore equivalent.

Deciding satisfiability via equivalence. The question whether φ is satisfiable can be decided by checking $\varphi \not\equiv \bot$. If it is not true, that φ evaluates to false under all models, then φ is satisfiable.

Deciding validity via 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 then φ is valid.

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

5.3 Normal Forms

As we have seen in **??**, SAT solver that implement the DPLL algorithm need the input to be a CNF. So far, we have seen two standard normal forms: the disjunctive normal form (DNF) and the conjunctive normal form (CNF). In the following, we will discuss how any propositional formula φ can be converted into a DNF(φ) or CNF(φ) via its truth table.

Disjunctive Normal Form (DNF)

For every row in the truth table under which φ evaluates to true, we form a conjunction of the literals. This means, variables which are assigned **T** are not negated and variables which are assigned **F** are negated. The resulting conjunctions are then connected with disjunctions to form DNF(φ).

Example 2

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*

Solution.

DNF(φ) $= (\neg a \wedge \neg b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge \neg c) \vee (\neg a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c)$.

Conjunctive Normal Form (CNF)

To convert the formula into a CNF, we consider the rows under which the formula evaluates to *false*. For every of these rows, we form a disjunction of the *negated literals*. This means, variables which are assigned **T** are negated and variables which are assigned **F** are not negated. The resulting disjunctions are connected via conjunctions to form CNF(φ).

Example 3

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

a	b	c	$\neg a \vee \neg(b \to c)$
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

Solution.
The resulting formula in DNF is

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

5.4 Tseitin Encoding

We want to use SAT solvers to check equivalence of two formulas φ_1 and φ_2 by deciding whether $\varphi_1 \oplus \varphi_2$ is not satisfiable. Using the truth table of $\varphi_1 \oplus \varphi_2$ to compute CNF($\varphi_1 \oplus \varphi_2$) is not a viable approach, since the resulting formula might by *exponentially* in size. Therefore, we do not generate the equivalent representation CNF($\varphi_1 \oplus \varphi_2$), 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 5.1 (Equisatisfiability.) 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 to apply *Tseitin transformation.* This transformation only results in a *linear* blow-up in the size of the formula.

The Tseitin Algorithm

Tseitin transformation takes the syntax tree for a propositional formula φ as input. The algorithm traverses the tree, beginning with the leaves, and associates a *auxilliary variable* to each node, i.e., to each subformula. We will call these auxilliary variables *tseitin variables.* By traversing the tree, the algorithm translates every subformula into a set of clauses. We will explain the Tseitin transformation for formulas that are restricted to the Boolean connectives $\land$, $\lor$, and $\neg$.

Step 1. Assign tseitin variables to all nodes in the parse tree, i.e. to each subformula.

Step 2. Add new clauses for each tseitin variable. Every tseitin variable represents either a $\neg$, $\land$, or $\lor$ node in the parse tree, i.e. a sub-formula of the form $\neg\varphi$, $\varphi \land \psi$, or $\varphi \lor \psi$. The *Tseitin-Rewriting Rules* define, which clauses are associated with which type of node:

For each node, let x be the tseitin variable, and let φ and ψ be the the two subformulas of the node.

- $\land$ node: for $x \leftrightarrow (\varphi \land \psi)$ introduce the clauses $(\neg x \lor \varphi) \land (\neg x \land \psi) \land (\neg\varphi \lor \neg\psi \lor x)$.
- $\lor$ node: for $x \leftrightarrow (\varphi \lor \psi)$ introduce the clauses $(\neg\varphi \lor x) \land (\neg\psi \lor x) \land (\neg x \lor \varphi \lor \psi)$.
- $\neg$ node: for $x \leftrightarrow \neg\varphi$ introduce the clauses $(\neg x \lor \neg\varphi) \land (\varphi \lor x)$.

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.

Example 4

Consider the formula $\varphi := ((p \lor q) \land r) \lor \neg p$. Compute an equisatisfiable CNF(φ) in CNF.

Solution. For each sub-formula, we introduce a tseitin variable.

$$\underbrace{\underbrace{((\underbrace{p \lor q}_{x_1}) \land r)}_{x_2} \lor \underbrace{\neg p}_{x_3}}_{x_\varphi}$$

Using the rewriting rules, we obtain the following sets of clauses and the

$$\begin{aligned}
\text{CNF}(\varphi) = \;& (\neg p \lor x_1) \land (\neg q \lor x_1) \land (\neg x_1 \lor p \lor q) \\
\land \;& (\neg x_2 \lor x_1) \land (\neg x_2 \lor r) \land (\neg x_1 \lor \neg r \lor x_2) \\
\land \;& (\neg x_3 \lor \neg p) \land (p \lor x_3) \\
\land \;& (\neg x_2 \lor x_\varphi) \land (\neg x_3 \lor x_\varphi) \land (\neg x_\varphi \lor x_2 \lor x_3) \\
\land \;& x_\varphi
\end{aligned}$$

Derivation of the Tseitin Rewriting Rules

In the following we derive how the clauses are generated for $\wedge$, $\vee$, and $\neg$ nodes.

- $x \leftrightarrow (p \wedge q)$ generates the clauses $(\neg x \vee p) \wedge (\neg x \vee q) \wedge (\neg p \vee \neg q \vee x)$ since:
$$\begin{array}{ll} x \leftrightarrow (p \wedge q) & \\ (x \rightarrow (p \wedge q)) \wedge ((p \wedge q) \rightarrow x) & \mid \varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\ (x \rightarrow p) \wedge (x \rightarrow q) \wedge ((p \wedge q) \rightarrow x) & \mid \varphi \rightarrow (\psi \wedge \chi) \equiv (\varphi \rightarrow \psi) \wedge (\varphi \rightarrow \chi) \\ (\neg x \vee p) \wedge (\neg x \wedge q) \wedge (\neg(p \wedge q) \vee x) & \mid \varphi \rightarrow \psi \equiv \neg\varphi \vee \psi \\ (\neg x \vee p) \wedge (\neg x \wedge q) \wedge (\neg p \vee \neg q \vee x) & \mid \neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi \end{array}$$
- $x \leftrightarrow (p \vee q)$ generates the clauses $(\neg p \vee x) \wedge (\neg q \vee x) \wedge (\neg x \vee p \vee q)$ since:
$$\begin{array}{ll} x \leftrightarrow (p \vee q) & \\ (x \rightarrow (p \vee q)) \wedge ((p \vee q) \rightarrow x) & \mid \varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\ (x \rightarrow (p \vee q)) \wedge (p \rightarrow x) \wedge (q \rightarrow x) & \mid \psi \vee \varphi \rightarrow \chi \equiv (\varphi \rightarrow \chi) \wedge (\psi \rightarrow \chi) \\ (\neg x \vee p \vee q) \wedge (\neg p \vee x) \wedge (\neg q \vee x) & \mid \varphi \rightarrow \psi \equiv \neg\varphi \vee \psi \\ (\neg p \vee x) \wedge (\neg q \vee x) \wedge (\neg x \vee p \vee q) & \mid \text{rearranging} \end{array}$$
- $x \leftrightarrow \neg p$ generates the clauses $(\neg x \vee \neg p) \wedge (p \vee x)$ since:
$$\begin{array}{ll} x \leftrightarrow \neg p & \\ (x \rightarrow \neg p) \wedge (\neg p \rightarrow x) & \mid \varphi \leftrightarrow \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\ (\neg x \vee \neg p) \wedge (\neg\neg p \vee x) & \mid \varphi \rightarrow \psi \equiv \neg\varphi \vee \psi \\ (\neg x \vee \neg p) \wedge (p \vee x) & \mid \neg\neg\varphi \equiv \varphi \end{array}$$

Properties of Tseitin Encoding

We make two observations about the set of clauses that is generated by the Tseitin transformation 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 an equisatisfiable CNF formula CNF(φ).
2. The constructed formula CNF(φ) is equisatisfiable to φ: CNF(φ) 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(φ) by simply dropping the additional variables that the algorithm has introduced.

5.5 CEC Example

We conclude this short chapter with a complete example.

Example 5

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.

Solution.

- We construct the formula φ:

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

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

$$\underbrace{\underbrace{\underbrace{(\underbrace{\neg a}_{x_1} \wedge \underbrace{\neg b}_{x_2})}_{x_4} \wedge \underbrace{\neg(\underbrace{\neg(\underbrace{a \vee b}_{x_3})}_{x_5})}_{x_6}}_{x_8} \vee \underbrace{\underbrace{\neg(\underbrace{\underbrace{\neg a}_{x_1} \wedge \underbrace{\neg b}_{x_2}}_{x_4})}_{x_7} \wedge \underbrace{\neg(\underbrace{a \vee b}_{x_3})}_{x_5}}_{x_9}}_{x_\varphi}$$

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

- Finally, $\mathrm{CNF}(\varphi)$ is given to a SAT solver. If the SAT solver determines that $\mathrm{CNF}(\varphi)$ is unsatisfiable, then $\varphi_1 \equiv \varphi_2$.

List of Definitions