

## Propositional Logic

This section introduces propositional logic. We will study syntax and model theoretic semantic of a language of classical propositional logic and we investigate various calculi for deciding certain properties of sentences in this language.

### Preliminaries

As a running example consider the simple digital circuit in figure 1 consisting of an or-gate (*or1*) and two inverters (*inv1* and *inv2*). The system description is given by the following propositional formulae, where the formulae are labelled in order to facilitate recognition of corresponding parts in figure 1. The mnemonic used in the names within the formulae is the following:

- The symbols  $\vee$ ,  $\wedge$  and  $\neg$  denote “or”, “and” and “not” respectively.
- *i1* and *i2* are denoting inputs in the or-gate, *i* an inverter and *o* denotes the output of a gate.
- *high* is a predicate which is intended to denote the fact, that the value specified by its arguments is “of high voltage”, i.e. is on.
- *ab* is a predicate which stands for “is abnormal”, i.e. the expression  $\neg(ab(inv1))$  can be read as “it is not the case, that the inverter one is not abnormal.”

$$\begin{aligned} OR1 : & \neg(ab(or1)) \rightarrow high(or1, o) \leftrightarrow (high(or1, i1) \vee high(or1, i2)) \\ INV1 : & \neg(ab(inv1)) \rightarrow high(inv1, o) \leftrightarrow \neg(high(inv1, i)) \\ INV2 : & \neg(ab(inv2)) \rightarrow high(inv2, o) \leftrightarrow \neg(high(inv2, i)) \\ CONN1 : & high(inv1, o) \leftrightarrow high(or1, i1) \\ CONN2 : & high(inv2, o) \leftrightarrow high(or1, i2) \end{aligned}$$

These formulae contain labels which express their intended meaning: OR1 stands for a formula describing the function of the or-gate, IN1 and IN2 are describing the (identical) behavior of the two inverters: CONN1 and CONN2 are specifying the way the inputs of the or-gate are connected with the outputs of the inverters. Note, that in this description the inputs and outputs of the system are not yet defined.

We observe from the graphical description that both inputs of the circuit have low voltage and the output also has low voltage, i.e. we could state this by the formulae  $\neg high(inv1, i)$ ,  $\neg high(inv2, i)$ ,  $\neg high(or1, o)$

This example demonstrates how a proposition like  $high(inv1, o)$  can be used to formally describe the behaviour of an electronic circuit. Note that  $high(inv1, o)$  can be read intuitively as ‘‘the output of inverter 1 is high’’, and of course this sentence can be further simplified by ignoring the phrase structure and by understanding it as a simple string: ‘‘the\_output\_of\_inverter\_1\_is\_high’’. If we would proceed with the other propositions in the same manner we would get formulae like this:

$$the\_output\_of\_inverter\_1\_is\_high \leftrightarrow the\_input\_of\_inverter\_1\_is\_high$$

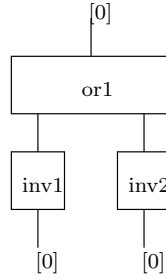


Figure 1: A simple electronic circuit

This formula consists of two propositions which are connected via a double arrow. It is easy to see that formulae get rather lengthy by this way and hence we can further abbreviate the propositions by simpler strings like  $P_1$  or  $P_2$ , which results in a formula  $P_1 \leftrightarrow P_2$ . It should be clear by the above argumentation that this formula carries for our purposes the same information as  $high(or1, i1) \leftrightarrow high(inv1, o)$ . The only difference is that with the latter form some intuition is associated with the reader – for our formal logical framework this is irrelevant and we will focus on arbitrary propositions.

In the following we will systematically avoid any intuition to real world meaning in the design of our language. We will focus solely on the study of the logical aspects, which can be expressed by our language and not to a certain intended meaning. In other words, our previous introduction of the logical formulae, which described the toy electronic circuit was too much oriented on an interpretation, which was triggered by the picture.

We will introduce the syntax of propositional logic by a very simple inductive definition and then we specify a formal semantics by another single definition.

## Syntax

For the definition of the syntax we have to assume a set of atomic formulae, which we take as the start of an inductive definition. With the help of three junctors and brackets as punctuation symbols we inductively define more complex formulae.

**Definition 1 (Syntax of propositional logic)** Assume a

- countable set of atomic formulae  $P_i$ , where  $i = 1, 2, 3, \dots$ ,
- the junctors  $\wedge, \vee$ , and  $\neg$ ,
- the punctuation symbols ( and ).

The set of propositional formulae is defined by the following induction:

- Atomic formulae are formula.
- If  $F$  and  $G$  are formulae, then  $(F \wedge G)$  and  $(F \vee G)$  are formulae.
- If  $F$  is a formula, then  $\neg F$  is a formula

The different kinds of formulae are called conjunction, disjunction and negation. Note, that this is just a convention about wording, until now, we didn't give any semantics to these junctors, which would justify these denotations.

This definition of the set of formulae also introduces a partial order in a very natural way, if we consider the concept of a subformula.

**Definition 2 (Subformula)** *A subformula of a formula is a substring, which is a formula.*

Note that the relation “is subformula” is indeed a partial order.

The set of formulae together with the subformula-relation is a well-founded relation.

In order to facilitate notations We introduce the following abbreviations:

$A, B, C$	instead of	$P_1, P_2, P_3$
$(F_1 \rightarrow F_2)$		$(\neg F_1 \vee F_2)$
$(F_1 \leftrightarrow F_2)$		$((F_1 \wedge F_2) \vee (\neg F_1 \wedge \neg F_2))$
$\bigvee_{i=1}^n F_i$		$(\dots((F_1 \vee F_2) \vee F_3) \vee \dots \vee F_n)$
$\bigwedge_{i=1}^n F_i$		$(\dots((F_1 \wedge F_2) \wedge F_3) \wedge \dots \wedge F_n)$

These notations (except the first one) are simple abbreviations; whenever those arrow-symbols or indexed junctors occur in a formulae, the corresponding subformula can be substituted according to this definition.

Coming back to our previous example you can see that

$$\neg(ab(or1)) \rightarrow high(or1, o) \leftrightarrow (high(or1, i1) \vee high(or1, i2))$$

can be written as a formula according to the above definition by introducing parenthesis:

$$(\neg(ab(or1)) \rightarrow (high(or1, o) \leftrightarrow (high(or1, i1) \vee high(or1, i2))))$$

The parenthesis are introduced in order to avoid any ambiguity. In order to increase readability, we will omit them whenever this is possible.

If we additionally apply the above abbreviation rules we would result in the following formula:

$$\begin{aligned} &(\neg\neg(ab(or1)) \vee \\ &\quad (high(or1, o) \wedge ((high(or1, i1) \vee high(or1, i2)) \vee \\ &\quad (\neg high(or1, o) \vee \neg(high(or1, i1) \vee high(or1, i2))))) \end{aligned}$$

## Semantics

**Definition 3 (Semantics of propositional logic)** *For the semantics of our formal language of propositions we do not refer to a specific intended interpretation. Moreover we only are interested in truth values. We assume an initial assignment of truth values to atomic formulae and based on this, we define the truth value of more complex formulae. The set of truth values is the set  $\{true, false\}$ .*

*An assignment for a set  $D$  of atomic formulae is a function*

$$\mathcal{A}|D \rightarrow \{true, false\}.$$

*Let  $E$  be the set of formulae containing  $D$ , i.e  $E \supseteq D$  and exactly those formulae which can be constructed from  $D$  according to the definition of the syntax. Then the extension of  $\mathcal{A}$  on  $E$ ,  $\mathcal{A}_E|E \rightarrow \{true, false\}$ , is given as:*

- $A \in D : \mathcal{A}_E(A) = \mathcal{A}(A)$
- $\mathcal{A}_E((F \wedge G)) = \begin{cases} true & \text{if } \mathcal{A}_E(F) = true \text{ and } \mathcal{A}_E(G) = true \\ false & \text{otherwise} \end{cases}$
- $\mathcal{A}_E((F \vee G)) = \begin{cases} true & \text{if } \mathcal{A}_E(F) = true \text{ or } \mathcal{A}_E(G) = true \\ false & \text{otherwise} \end{cases}$
- $\mathcal{A}_E(\neg F) = \begin{cases} true & \text{if } \mathcal{A}_E(F) = false \\ false & \text{otherwise} \end{cases}$

In the following we will omit the index  $E$  to indicate the extension of an assignment  $\mathcal{A}$ . Note that this is the right place to name formulae of type  $(F \wedge G)$  as conjunctions, formulae of type  $(F \vee G)$  as disjunctions and formulae of type  $\neg F$  as negations. The following is an example evaluation by means of the definition of  $\mathcal{A}$ : Assume as given  $\mathcal{A}(A) = true$ ,  $\mathcal{A}(B) = true$  and  $\mathcal{A}(C) = false$ , then

$$\begin{aligned}
\mathcal{A}(\neg((A \wedge B) \vee C)) &= \begin{cases} true & \text{if } \mathcal{A}(((A \wedge B) \vee C)) = false \\ false & \text{otherwise} \end{cases} \\
&= \begin{cases} false & \text{if } \mathcal{A}(((A \wedge B) \vee C)) = true \\ true & \text{otherwise} \end{cases} \\
&= \begin{cases} false & \text{if } \mathcal{A}(((A \wedge B))) = true \text{ or } \mathcal{A}(C) = true \\ true & \text{otherwise} \end{cases} \\
&= \begin{cases} false & \text{if } \mathcal{A}(((A \wedge B))) = true \\ true & \text{otherwise} \end{cases} \\
&= \begin{cases} false & \text{if } \mathcal{A}(A) = true \text{ and } \mathcal{A}(B) = true \\ true & \text{otherwise} \end{cases} \\
&= false
\end{aligned}$$

**Definition 4** Let  $\mathcal{A}$  be an assignment and  $F$  a formula.  $\mathcal{A}$  is called assignment for  $F$ , if  $\mathcal{A}$  is defined for every subformula of  $F$ .

If  $\mathcal{A}$  is an assignment for  $F$  and  $\mathcal{A}(F) = true$  we call  $\mathcal{A}$  a model for  $F$  and we write  $\mathcal{A} \models F$

If  $\mathcal{A}$  is an assignment for  $F$  and  $\mathcal{A}(F) = false$ , we write  $\mathcal{A} \not\models F$ .

A formula  $F$  is called satisfiable, iff there is a model for  $F$ , otherwise  $F$  is called unsatisfiable.  $F$  is called valid (or a tautology) iff every assignment for  $F$  is a model. We write  $\models F$  resp.  $\not\models F$ .

**Theorem 1** A formula  $F$  is valid, iff  $\neg F$  is unsatisfiable.

**Proof:**

$F$  is a tautology

iff every assignment for  $F$  is a model for  $F$

iff every assignment for  $F$  (which is of course, also an assignment for  $\neg F$ ) is not a model

for  $\neg F$   
iff  $\neg F$  has no model  
iff  $\neg F$  is unsatisfiable.

**Definition 5 (Consequence)** A formula  $G$  is called a consequence of the formulae  $F_1, \dots, F_k$ , if for every assignment  $\mathcal{A}$  for  $G$  and  $F_1, \dots, F_k$  holds: if  $\mathcal{A}$  is a model for  $F_1, \dots, F_k$ , then  $\mathcal{A}$  is a model for  $G$ . We write  $\{F_1, \dots, F_k\} \models G$ .

The following theorem is a simple consequence from definitions:

**Theorem 2**  $G$  is called a consequence of the formulae  $F_1, \dots, F_k$ ,  
iff  $((\bigwedge_{i=1}^k F_i) \rightarrow G)$  is a tautologie  
iff  $((\bigwedge_{i=1}^k F_i) \wedge \neg G)$  is unsatisfiable.

Obviously the validity of a formula depends only of the assignments for its atomic subformulae: If  $\mathcal{A}$  and  $\mathcal{A}'$  are assignments for  $F$  and if they yield the same value for every occurring atomic subformulae, then  $\mathcal{A}(F) = \mathcal{A}'(F)$  holds. As a consequence we can state, that it suffices for a test of the validity of a formula  $F$  to check the infinitely many assignments of its atomic subformulae.

Such a check can be depicted easily in a tableau of the following form, where  $F$  is an arbitrary formula, containing  $n$  distinct atomic formulae  $A_i$ .

	$A_1$	$\dots$	$A_{n-1}$	$A_n$	$F$
$\mathcal{A}_1$	false	$\dots$	false	false	$\mathcal{A}_1(F)$
$\mathcal{A}_2$	false	$\dots$	false	true	$\mathcal{A}_2(F)$
$\mathcal{A}_{2n}$	true	$\dots$	true	true	$\mathcal{A}_n(F)$

When applying this procedure it might be helpful to introduce intermediate results for subformulae, as done in the following example.

A	B	$\neg A$	$A \rightarrow B$	$(\neg A \rightarrow (A \rightarrow B))$
false	false	true	true	true
false	true	true	true	true
true	false	false	false	true
true	true	false	true	true

Note that we just have depicted an algorithm to check the validity of a formula. Assume the Formula contains  $n$  atomic subformulae, then our just constructed tableaux contain  $2^n$  lines. To estimate the costs for such an exponential algorithm, assume that the computation of one line takes 1 micro-second. Is  $F$  contains only 100 atomic subformulae the computation of the tableau would take  $2^{100}$  micro-seconds.

## Interaction: Truth Tables

This is your set of current propositional formulas:

---

Currently empty.

---

**Current formulas manipulation:**

[Add](#)

random formulas of complexity  
low      medium      high

Enter some formulas:

Help:

[Add to](#) / [Overwrite](#) current formulas

[Delete selected](#) / [Delete all](#) current formulas

**Save/Reload current formulas:**

[Save](#) to database (enter name):

[Reload](#) from database (select name):

[Reload](#) from file (enter file name):

---

[Generate truth table\(s\) for \*selected\* formulas](#)    [Delete current truth tables](#)



And here are the resulting truth tables (possibly from a previous run):    Currently empty.

□

The problem whether a propositional formula is satisfiable is called SAT and the corresponding tautology problem is called TAUT.

SAT is an NP-complete problem, and hence it is not known whether SAT is tractable or not. Whether TAUT is in NP is still an open problem. For TAUT we know, that it is in NP iff NP is closed under complementation. SAT and TAUT, both, play a prominent role in the study of complexity theory, in particular with respect to the question whether  $P = NP$ .

**Problem 1**

Compute truth tables for the following formulae. Decide for each formula whether it is valid (a tautology), satisfiable or unsatisfiable.

1.  $(A \wedge B) \wedge (B \rightarrow C)$
2.  $(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$
3.  $(\neg A \vee \neg(\neg B \vee A)) \vee A$
4.  $(A \rightarrow (B \wedge C)) \leftrightarrow ((A \rightarrow B) \wedge (A \rightarrow C))$
5.  $(A \vee (B \wedge A)) \wedge ((C \vee \neg C) \rightarrow \neg A)$
6.  $((C \vee B) \wedge (C \vee \neg B)) \wedge \neg C$
7.  $\neg(\neg A \vee \neg(\neg B \vee \neg A))$
8.  $(A \rightarrow (B \rightarrow A))$
9.  $(A \vee (B \wedge A)) \wedge ((C \vee \neg C) \rightarrow \neg A)$
10.  $((A \vee \neg B) \vee ((\neg A \wedge \neg C) \wedge B)) \leftrightarrow ((A \vee \neg C) \vee \neg B)$
11.  $\neg(A \wedge \neg(B \wedge \neg C))$

**Problem 2**

"What is the secret of your long life?" a 100-year old man was asked. He answered: "I apply the following diet rules very strictly: If I drink no wine at dinner then I have always fish. Whenever I take fish and wine for dinner, it is without garlic. If I have garlic or wine I avoid fish"

1. Formalize these rules with the help of propositional logic.
2. Try to simplify the advice of the 100-year old man.

**Problem 3**

Define the following functions recursively by induction over the construction of propositional formulae:

1.  $\alpha(F)$ : Set of atomic formula in  $F$
2.  $\beta(F)$ : Number of the binary junctors  $\wedge$  and  $\vee$  in  $F$

**Note:** For

$$F = (\neg(A \vee B) \vee C) \wedge ((\neg A \vee B) \wedge C)$$

$\alpha(F) = \{A, B, C\}$  and  $\beta(F) = 5$  holds.

**Problem 4**

Given the formulae  $F_n = A_1 \dot{\vee} \cdots \dot{\vee} A_n$ , in which  $A_1, \dots, A_n$  are atomic formulae ( $n \geq 1$ ) where  $\dot{\vee}$  denotes exclusive or. Prove for all  $n \in \mathbb{N}$ : A suitable assignment  $\mathcal{A}$  for  $F_n$  is a model for  $F_n$  (i.e.  $\mathcal{A} \models F_n$ ) iff  $\mathcal{A}(A_i) = 1$  holds for an odd number of  $A_i$  ( $1 \leq i \leq n$ ).

**Problem 5**

In the following we investigate formulae, in which only atoms  $A_1, \dots, A_n$  occur.

1. How many of such formulae with different truth tables exist at most?
2. Is there for every truth table a formula? If yes, please indicate a construction!

**Problem 6**

If the colonel was not in the room during the murder then he cannot know the weapon of the murderer. The butler lies or he knows the murderer. If Lady Barntree is the murderer then the colonel was in the room during the murder or the butler lies. The butler knows the murderer or the colonel was not in the room during the murder. The policeman concludes that Lady Barntree is the murderer.

Give propositional variables for every statement of the argumentation. Write the argumentation as a set  $M$  of propositional formulae of the prerequisites and the conclusion as a propositional formula  $F$ .

**Problem 7**

Formalize the following expressions and then simplify the corresponding formulae and the verbal propositions.

1. It is not true that his mother is English and his father French.
2. It is not true that he is studying physics but not mathematics.
3. It is not true that the wages are going down and the prices are going up.

4. It is not true that it is not cold or rainy.

### Problem 8

The professor proposes to make a new conception for the lunch in the University restaurant:

1. There must be bread with every lunch if there is no dessert.
2. If bread and dessert are served then there is no soup.
3. If soup is served then there is no bread.

Help the management to fulfill the wishes of the professor. For this

1. formalize the three propositions (as implications, disjunctions/conjunctions) and combine them into **one** logical formula.
2. Give a truth table for this logical formula. Is there a model for the formula?
3. Give a colloquial verbalization for the assignment.

## Equivalence and Normal Forms

Until now, we only discussed single formulae and their semantical properties. In this section we start investigating whether formulae can be transformed into another form, without changing their semantics. For this we introduce the concept of *logical equivalence*, which can be used to investigate the transformation of a given formula into its *normal form*.

**Definition 6** *The formulae  $F$  and  $G$  are called (semantically) equivalent, iff for all assignments  $\mathcal{A}$  for  $F$  and  $G$ ,  $\mathcal{A}(F) = \mathcal{A}(G)$ . We write  $F \equiv G$ .*

Formulae containing different subformulae can be equivalent, e.g. all tautologies are equivalent.

More interesting is the following theorem:

**Theorem 3 (Substitutivity)** *Let  $F \equiv G$  and  $H$  a formula which contains at least one occurrence of the subformula  $F$ . Then it holds  $H \equiv H'$ , where  $H'$  is obtained from  $H$  by substituting any occurrence of  $F$  by  $G$ .*

**Proof:** The proof is by induction over the structure of the subformula  $H$ :

Assume for the induction start, that  $H$  is atomic, hence  $H = F$  holds, and the result of substituting the only occurrence of  $F$  by  $G$  results in  $H' = G$  and because  $F \equiv G$ , we have  $H \equiv H'$ .

Assume the theorem holds for all proper subformulae of  $H$ : If  $F = H$  we have the same argumentation as above in the start of the induction. If  $F \neq H$  we have three cases:

- $H = \neg H_1$ : Because  $H_1$  is a subformula of  $H$  we can conclude that  $H_1 \equiv H'_1$ , where  $H'_1$  is constructed by substituting any occurrence of  $F$  by  $G$ . From the definition of the semantics of  $\neg$  we conclude that  $\neg H_1 \equiv \neg H'_1$  and hence  $H \equiv H'_1$ .

- $H = (H_1 \vee H_2)$ : Assume without loss of generality, that  $F$  occurs in  $H_1$ . Then, again we can conclude from the induction assumption, that  $H_1 \equiv H'_1$  and from the definition of the semantics of  $\vee$  we conclude, that  $(H_1 \vee H_2) = (H'_1 \vee H_2)$ .
- $H = (H_1 \wedge H_2)$  is similar.

**Theorem 4** *The following equivalences hold:*

$$\begin{array}{ll}
 (F \wedge F) \equiv F & \\
 (F \vee F) \equiv F & \text{(Idempotence)} \\
 \\
 (F \wedge G) \equiv (G \wedge F) & \\
 (F \vee G) \equiv (G \vee F) & \text{(Commutativity)} \\
 \\
 ((F \wedge G) \wedge H) \equiv (F \wedge (G \wedge H)) & \\
 ((F \vee G) \vee H) \equiv (F \vee (G \vee H)) & \text{(Associativity)} \\
 \\
 (F \wedge (F \vee G)) \equiv F & \\
 (F \vee (F \wedge G)) \equiv F & \text{(Absorption)} \\
 \\
 (F \wedge (G \vee H)) \equiv ((F \wedge G) \vee (F \wedge H)) & \\
 (F \vee (G \wedge H)) \equiv ((F \vee G) \wedge (F \vee H)) & \text{(Distributivity)} \\
 \\
 \neg \neg F \equiv F & \text{(Double negation)} \\
 \\
 \neg(F \wedge G) \equiv (\neg F \vee \neg G) & \\
 \neg(F \vee G) \equiv (\neg F \wedge \neg G) & \text{(deMorgan's rule)} \\
 \\
 (F \vee G) \equiv F, \text{ if } F \text{ is a tautology} & \\
 (F \wedge G) \equiv G, \text{ if } F \text{ is a tautology} & \text{(Rule of Tautology)} \\
 \\
 (F \vee G) \equiv G, \text{ if } F \text{ is unsatisfiable} & \\
 (F \wedge G) \equiv F, \text{ if } F \text{ is unsatisfiable} & \text{(Rule of Unsatisfiability)}
 \end{array}$$

**Proof:** All equivalences can be proved by truth tables. This is done here for the second rule of absorption:

$F$	$G$	$(F \wedge G)$	$(F \vee (F \wedge G))$
false	false	false	false
false	true	false	false
true	false	false	true
true	true	true	true

Let us now use these equivalences together with the theorem of substitutivity (TS) to prove the following equivalence:

$$((A \vee (B \vee C)) \wedge (C \vee \neg A)) \equiv ((B \wedge \neg A) \vee C)$$

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

$\equiv ((A \vee B) \vee C) \wedge (C \vee \neg A)$	Associativity and TS
$\equiv ((C \vee (A \vee B)) \wedge (C \vee \neg A))$	Commutativity and TS
$\equiv (C \vee ((A \vee B) \wedge \neg A))$	Distributivity
$\equiv (C \vee (\neg A \wedge (A \vee B)))$	Commutativity and TS
$\equiv (C \vee ((\neg A \wedge A) \vee (\neg A \wedge B)))$	Distributivity and TS
$\equiv (C \vee (\neg A \wedge B))$	Unsatisfiability and TS
$\equiv (C \vee (B \wedge \neg A))$	Commutativity and TS
$\equiv ((B \wedge \neg A) \vee C)$	Commutativity and TS

**Problem 1**

The binary junctor  $\dot{\vee}$  for exclusive disjunction is defined by  $F \dot{\vee} G = F \leftrightarrow (\neg G)$ . Show that the following holds for propositional logic formulae  $F$ ,  $G$  and  $H$ :

1.  $F \dot{\vee} G \equiv G \dot{\vee} F$  (Commutativity).
2.  $(F \dot{\vee} G) \dot{\vee} H \equiv F \dot{\vee} (G \dot{\vee} H)$  (Associativity).

**Problem 2**

Show the following by equivalence transformaton. Give to all remodelling steps the used rules!

1.  $(A \rightarrow (B \vee C)) \equiv (A \rightarrow B) \vee (A \rightarrow C)$
2.  $(A \rightarrow B) \rightarrow A \equiv (B \rightarrow A) \wedge (B \vee A)$
3.  $A \leftrightarrow \neg B \equiv \neg(A \leftrightarrow B)$
4.  $(A \rightarrow (B \wedge C)) \equiv (A \rightarrow B) \wedge (A \rightarrow C)$
5.  $(A \rightarrow B) \rightarrow (B \rightarrow A) \equiv B \rightarrow A$
6.  $(A \vee \neg B) \vee ((\neg A \wedge \neg C) \wedge B) \equiv (A \vee \neg C) \vee \neg B$

**Problem 3**

Prove with equivalences:

1.  $p \vee \neg(p \wedge q)$  is a tautology.
2.  $(p \vee \neg q) \wedge (\neg p \wedge q)$  is unsatisfiable.

**Problem 4**

The binary junctor  $\downarrow$  with the meaning *neither-nor* is defined by  $F \downarrow G = \neg(F \vee G)$ . Let  $F$  be a propositional logic formula, which contains only the operators  $\wedge$ ,  $\vee$  and  $\neg$ . Prove that for every formula  $F$  a formula  $\mathcal{T}(F)$  exists, such that  $F \equiv \mathcal{T}(F)$  and  $\mathcal{T}(F)$  is built by using only the junctor  $\downarrow$ .

**Problem 5**

Let  $F$  be a propositional logic formula, which contains only the operators  $\wedge$ ,  $\vee$  and  $\neg$ . Prove that for every formula  $F$  a formula  $\mathcal{T}(F)$  exists, such that  $F \equiv \mathcal{T}(F)$  and  $\mathcal{T}(F)$  is built by using only the junctors  $\rightarrow$  and  $\dot{\vee}$ .

**Problem 6**

The following formulae are given.

$$A \equiv D \vee B \quad B \equiv \neg B \vee \neg D \vee \neg S \quad C \equiv (\neg S \wedge D) \vee \neg B$$

1. Simplify  $A \wedge B \wedge C$  by using  $B \wedge C \equiv C$ .
2. Simplify  $A \wedge B \wedge C$  by using equivalences but not  $B \wedge C \equiv C$ .

**Definition 7 (Normal Forms)** A literal is an atomic formula or the negation of an atomic formula (positive or negative, resp.)

A formula  $F$  is in conjunctive normalform (CNF) iff

$$F = \left( \bigwedge_{i=1}^n \left( \bigvee_{j=1}^{m_i} L_{i,j} \right) \right)$$

where  $L_{ij}$  is a literal.

A formula  $F$  is in disjunctive normalform (DNF) iff

$$F = \left( \bigvee_{i=1}^n \left( \bigwedge_{j=1}^{m_i} L_{i,j} \right) \right)$$

where  $L_{ij}$  is a literal

**Theorem 5** For every formula  $F$  there is an equivalent formula which is in DNF and an equivalent formula which is in CNF.

Let us formulate an algorithm to transform a given formula  $F$  into an equivalent normalform:

**Given:** A formula  $F$

1. Substitute in  $F$  every subformula of the form

$$\begin{aligned} \neg \neg G & \text{ by } G \\ \neg (G \wedge H) & \text{ by } (\neg G \vee \neg H) \\ \neg (G \vee H) & \text{ by } (\neg G \wedge \neg H) \end{aligned}$$

until there is no subformula of this kind.

2. Substitute in the result from the above step every subformula of the form

$$\begin{aligned} (F \vee (G \wedge H)) & \text{ by } ((F \vee G) \wedge (F \vee H)) \\ ((F \wedge G) \vee H) & \text{ by } ((F \vee H) \wedge (G \vee H)) \end{aligned}$$

until there is no subformula of this kind.

**Result:** An equivalent formula in CNF

## Interaction: Transformation into CNF

This is your set of current propositional formulas:

---

Currently empty.

---

### Current formulas manipulation:

[Add](#) random formulas of complexity  
low medium high

Enter some formulas: Help:

[Add to](#) / [Overwrite](#) current formulas

[Delete selected](#) / [Delete all](#) current formulas


### Save/Reload current formulas:

[Save](#) to database (enter name):

[Reload](#) from database (select name):

[Reload](#) from file (enter file name):

---

[Convert selected](#) / [Convert all](#) formulas to CNF, replacing the current clauses. 

And this the resulting clause set (possibly from a previous run):

---

Currently empty.

---

## Exercises

This is your set of current propositional formulas:

---

Currently empty.

---

### Current formulas manipulation:

[Add](#) random formulas of complexity  
low medium high

Enter some formulas: Help:

[Add to](#) / [Overwrite](#) current formulas

[Delete selected](#) / [Delete all](#) current formulas

### Save/Reload current formulas:

[Save](#) to database (enter name):

[Reload](#) from database (select name):

[Reload](#) from file (enter file name):

## Perform this exercise with the selected propositional formula

### Current exercise:

Transform the following formula into CNF:

Currently empty. How to do the exercise:

### Current transformations:

Currently empty.

[Apply](#) next formula:

[Help](#):

Until now, we investigate the transformation of a propositional formula into an equivalent normal form. Another problem in the context of normal forms is, to construct a normal form formula from a given truth table; i.e. the formula itself is not known, but its behaviour is given by a truth table. Let's read a normalform formula from a truth table: Assume a formula  $F$ , which is given by the following truth table.



A	B	C	F
false	false	false	true
false	false	true	false
false	true	false	false
false	true	true	false
true	false	false	true
true	false	true	true
true	true	false	false
true	true	true	false

In order to construct a formula in DNF, which is equivalent to  $F$ , we have to take into account, that every line of the table which yields the truthvalue *true* gives one conjunction: if the assignment of the literal  $A_i$  is *true* it is included as  $\dots \wedge A_i \wedge \dots$ , if is *false* we include  $\dots \wedge \neg A_i \wedge \dots$ . For the above example we get as a DNF:

$$\begin{aligned} &(\neg A \wedge \neg B \wedge \neg C) \vee \\ &(A \wedge \neg B \wedge \neg C) \vee \\ &(A \wedge \neg B \wedge C) \end{aligned}$$

If we change in the above procedure the roles of *true* and *false* and  $\vee$  and  $\wedge$  we arrive at a CNF:

$$\begin{aligned} &(A \vee B \vee \neg C) \wedge \\ &(A \vee \neg B \vee C) \wedge \\ &(A \vee \neg B \vee \neg C) \wedge \\ &(\neg A \vee \neg B \vee C) \wedge \\ &(\neg A \vee \neg B \vee \neg C) \end{aligned}$$

We introduce a special representation for formulae in normalform. In our circuit-example from the introduction we already used a very special form of normalforms, namely the implication form for formulae in CNF: every subformula  $F_i$  of a conjunction  $F_1, \dots, F_l$  is written as:

$$A_1 \wedge \dots \wedge A_n \rightarrow B_1 \vee \dots \vee B_m$$

It is easy to see, that this implication is logically equivalent to a disjunction  $\neg A_1 \vee \dots \vee \neg A_n \vee B_1 \vee \dots \vee B_m$ . Sometimes the implication is written as

$$A_1, \dots, A_n \rightarrow B_1; \dots; B_m$$

Even the following ambiguous notation is used in some cases, where the comma in the premiss stands for a conjunction and the comma in the conclusion for a disjunction:

$$A_1, \dots, A_n \rightarrow B_1, \dots, B_m$$

For some important procedures for logical reasoning it is mandatory to represent the formulae not only in one of the above notations for a normalform, moreover, it is necessary to use the so-called clause-form from the following definition.

**Definition 8** *If  $F$  is a formula in CNF, i.e.*

$$F = (L_{1,1} \vee \dots \vee L_{1,n_1}) \wedge \dots \wedge (L_{k,1} \vee \dots \vee L_{k,n_k})$$

*then its corresponding representation in clause form is given as*

$$F = \{\{L_{1,1}, \dots, L_{1,n_1}\}, \dots, \{L_{k,1}, \dots, L_{k,n_k}\}\}$$

*The sets  $\{L_{i,1}, \dots, L_{i,n_i}\}$  are called clauses.*

This representation as sets of literals has the advantage that literals occur in no special order and that multiple occurrences of a literal in a disjunction are “merged” in its clause form.

Note that as a consequence we have built in associativity, commutativity and Idempotence into the representation.

$$\begin{array}{ll} ((A_1 \vee \neg A_2) \wedge (A_3 \wedge A_3)) & \{\{A_1, \neg A_2\}, \{A_3\}\} \\ (A_3 \wedge (\neg A_2 \vee A_1)) & \\ \vdots & \end{array}$$

**Problem 1**

Create formulae in (a) conjunctive or (b) disjunctive normal form which are equivalent to:

$$A \dot{\vee} B \dot{\vee} C$$

where  $\dot{\vee}$  denotes exclusive or.

**Problem 2**

The theorem prover OTTER uses the following optimization rules for clause sets:

**Subsumption:** If the literals of a clause  $K$  are a subset of an another clause  $K'$  remove  $K'$  from the set of clauses.

**Deleting by unit clause:** If the set of clauses contains a unit clause  $L$  -here  $L$  is single literal unit-clause- every occurrence of a complementary literal  $\bar{L}$  in a clause is deleted.

Which laws of the logic justify these procedures?

**Problem 3**

Generate truth tables for the following formulae. Give the conjunctive and disjunctive normal forms of the formulae. Which one of the relations  $F \models G$ ,  $G \models F$ ,  $F \equiv G$  and  $F = G$  holds?

1.  $F = (A \wedge \neg(B \vee C)) \dot{\vee} ((A \rightarrow B) \vee (A \rightarrow C),)$  wobei  $F \dot{\vee} G = F \leftrightarrow (\neg G)$  gilt.
2.  $G = ((A \vee (B \wedge A)) \rightarrow \neg A) \vee C$

**Problem 4**

Generate a CNF and a DNF form the following truth table for the formula  $F$ .

A	B	C	F
0	0	0	0
1	0	0	1
0	1	0	0
1	0	0	1
1	1	0	1
1	0	1	0
0	1	1	1
1	1	1	0

**Problem 5**

Generate a CNF from the formula  $(P \wedge (Q \rightarrow R) \rightarrow S)$

**Horn clauses**

In this subsection we introduce a special class of formulae which are of particular interest for logic programming. Furthermore it turns out that these formulae admit an efficient test for satisfiability.

**Definition 9** *A formula is a Horn formula if it is in CNF and every disjunction contains at most one positive literal. Horn clauses are clauses, which contain at most one positive literal.*

$$\begin{aligned}
 F = & (A \vee \neg B) && \wedge && (\neg C \vee \neg A \vee D) \\
 & \wedge (\neg A \vee \neg B) && \wedge && D \wedge \neg E \\
 F \equiv & (B \rightarrow A) && \wedge && (C \wedge A \rightarrow D) \\
 & \wedge (A \wedge B \rightarrow \text{false}) && \wedge && (\text{true} \rightarrow D) \wedge (E \rightarrow \text{false})
 \end{aligned}$$

where *true* is a tautology and *false* is an unsatisfiable formula.  
 In clause form this can be written as

$$\{A, \neg B\}, \{\neg C, \neg A, D\}, \{(\neg A, \neg B)\}, \{D\}, \{\neg E\}$$

and in the context of logic programming this is written as:

$$\left\{ \begin{array}{l} A \leftarrow B \\ D \leftarrow A \\ \quad \leftarrow A, B \\ \quad \leftarrow E \\ D \leftarrow \quad \quad \quad \end{array} \right\}$$

For Horn formula there is an efficient algorithm to test satisfiability of a formula  $F$  :

Deciding Satisfiability of Horn Formulae

1. If there is a subformula of the kind  $true \rightarrow A$  label every occurrence of  $A$  in  $F$ .
2. Apply the following rules until none of them is applicable:
  - If  $A_1 \wedge \dots \wedge A_n \rightarrow B$  is a subformula and  $A_1 \dots A_n$  are all labeled and  $B$  is not labeled then label every occurrence of  $B$  in  $F$ .
  - If  $A_1 \wedge \dots \wedge A_n \rightarrow false$  is a subformula and  $A_1 \dots A_n$  are all labeled then **Stop: Unsatisfiable**
3. **Stop: Satisfiable** The assignment  $\mathcal{A}(A) = true$  iff  $A$  is labeled is a model.

**Theorem 6** *The above algorithm is correct and stops after  $n$  steps, where  $n$  is the number of atoms in the formula.*

As an immediate consequence we see, that a Horn formula is satisfiable if there is no subformula of the form  $A_1 \wedge \dots \wedge A_n \rightarrow false$ .

Horn formulae admit least models, i.e.  $\mathcal{A}$  is a least model for  $F$  if for every model  $\mathcal{A}'$  and for every atom  $B$  in  $F$  holds: if  $\mathcal{A}'(B) = true$  then  $\mathcal{A}(B) = true$  . Note, that this least model property does not hold for non Horn formulae: as an example take  $A \vee B$  which is obviously non Horn.  $\mathcal{A}_1(A) = true, \mathcal{A}_1(B) = false$  is a least model and  $\mathcal{A}_2(A) = false, \mathcal{A}_2(B) = true$  as well, hence we have two least models.

### Labeling algorithm

This is your set of current propositional clauses:

---

Currently empty.

---

<b>Current clauses manipulation:</b>	
Enter some clauses: <input style="width: 90%;" type="text"/>  <a href="#">Add to</a> / <a href="#">Overwrite</a> current clauses	Help: <input style="width: 90%;" type="text"/>  <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> <a href="#">Delete selected</a> / <a href="#">Delete all</a> current Clauses         </div> <div style="border: 1px solid black; padding: 5px;"> <a href="#">Convert selected</a> / <a href="#">Convert all</a> current propositional formulas to CNF, replacing the current clauses.         </div>
<b>Save/Reload current clauses:</b>	
<a href="#">Save</a> to database (enter name): <input style="width: 90%;" type="text"/>	<a href="#">Reload</a> from database (select name): <input style="width: 90%;" type="text"/>  <a href="#">Reload</a> from file (enter file name): <input style="width: 90%;" type="text"/>

[Apply labeling algorithm to the current clauses](#)

[Delete current labeling algorithm result](#)

This is the result, (possibly from a previous run):

Currently none. □

**Problem 1**

Let  $F$  be a propositional logical formulae and  $S$  a subset atomic formula occurring in  $F$ . Let  $T_S(F)$  be the formula which results from  $F$  by replacing all occurrences of an atomic formulae  $A \in S$  by  $\neg A$ . Example:  $T_{\{A\}}(A \wedge B) = \neg A \wedge B$ .

Prove or disprove: There exists an  $S$  for

1.  $F = A \dot{\vee} B$  or
2.  $F = A \dot{\vee} B \dot{\vee} C$ ,

so that  $T_S(F)$  is equivalent to a Horn formula  $H$  (i.e.  $T_S(F) \equiv H$ ).

**Problem 2**

Apply the marking algorithm to the following formula  $F$ . Which is a least model?

$$(A \vee \neg D \vee \neg E) \wedge (B \vee \neg C) \wedge (\neg B \vee \neg D) \wedge D \wedge (\neg D \vee E)$$

1.  $F = (\neg A \vee \neg B \vee \neg C) \wedge \neg D \wedge (\neg E \vee A) \wedge E \wedge B \wedge (\neg F \vee C) \wedge F$
2.  $F = (A \vee \neg D \vee \neg E) \wedge (B \vee \neg C) \wedge (\neg B \vee \neg D) \wedge D \wedge (\neg D \vee E)$

**Problem 3**

Decide which one of the indicated CNFs are Horn formulae and transform then into a conjunction of implications:

1.  $(A \vee B \vee C) \wedge (A \vee \neg C) \wedge (\neg A \vee B) \wedge \neg B$
2.  $(S \vee \neg P \vee Q) \wedge (S \vee \neg P \vee \neg R)$
3.  $(A \vee \neg A)$
4.  $A \wedge (\neg A \vee B) \wedge (\neg B \vee \neg C \vee D) \wedge (\neg E) \wedge (\neg A \vee \neg C) \wedge D$

## Resolution

In this subsection we will develop a calculus for propositional logic. Until now we have a language, i.e. a set of formulae and we have investigated into semantics and some properties of formulae or sets of clauses. Now we will introduce an inference rule, namely the resolution rule, which allows to derive new clauses from given ones.

**Definition 10** A clause  $R$  is a resolvent of clauses  $C_1$  and  $C_2$ , iff there is a literal  $L \in C_1$  and  $\bar{L} \in C_2$  and

$$R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$$

$$\text{where } \bar{L} = \begin{cases} \neg A & \text{if } L = A \\ A & \text{if } L = \neg A \end{cases}$$

Note that there is a special case, if we construct the resolvent out of two literals, i.e.  $L$  and  $\bar{L}$  can be resolved upon and yield the empty set. This empty resolvent is depicted by the special symbol  $\square$ .

$\square$  denotes an unsatisfiable formula. We define a clause set, which contains this empty clause to be unsatisfiable,

In the following we investigate in properties of the resolution rule and the entire calculus. The following Lemma is stating the correctness of one single application of the resolution rule.

**Theorem 7** If  $S$  is a set of clauses and  $R$  a resolvent of  $C_1, C_2 \in S$ , then

$$S \equiv S \cup \{R\}$$

**Proof:** Let  $\mathcal{A}$  be an assignment for  $S$ ; hence it is an assignment for  $S \equiv S \cup \{R\}$  as well. Assume  $\mathcal{A} \models S$ : hence for all clauses  $C$  from  $S$ , we have that  $\mathcal{A} \models C$ . The resolvent  $R$  of  $C_1$  and  $C_2$  looks like:

$$R = (C_1 - \{L\}) \cup (C_2 - \{\bar{L}\})$$

where  $L \in C_1$  and  $\bar{L} \in C_2$ . Now there are two cases:

- $\mathcal{A} \models L$ : From  $\mathcal{A} \models C_2$  and  $\mathcal{A} \not\models \bar{L}$ , we conclude  $\mathcal{A} \models (C_2 - \{\bar{L}\})$  and hence  $\mathcal{A} \models R$ .
- $\mathcal{A} \not\models L$ : From  $\mathcal{A} \models C_1$  we conclude  $\mathcal{A} \models (C_1 - \{L\})$  and hence  $\mathcal{A} \models R$ .

The opposite direction of the lemma is obvious.

**Definition 11** Let  $S$  be a set of clauses and

$$Res(S) = S \cup \{R \mid R \text{ is a resolvent of two clauses in } S\}$$

then

$$\begin{aligned} Res^0(S) &= S \\ Res^{n+1}(S) &= Res(Res^n(S)), n \geq 0 \\ Res^*(S) &= \bigcup_{n \geq 0} Res^n(S) \end{aligned}$$

If we understand the process of iterating the *Res*-operator as a procedure for deriving new clauses from a given set, and in particular to derive possibly the empty clause, we have to ask, under which circumstances we get the empty clause, and vice versa, what does it mean if we get it. These properties are investigated in the following two Theorems.

**Theorem 8 (Correctness)** *Let  $S$  be a set of clauses. If  $\square \in Res^*(S)$  then  $S$  is unsatisfiable.*

**Proof:** From  $\square \in Res^*(S)$  we conclude, that  $\square$  is obtained by resolution from two clauses  $C_1 = \{L\}$  and  $C_2 = \{\bar{L}\}$ . Hence there is a  $\exists n \geq 0$  such that  $\square \in Res^n(S)$  and  $C_1, C_2 \in Res^n(S)$  and therefore  $Res^n(S)$  is unsatisfiable. From Theorem 7 we conclude that  $Res^n(S) \equiv S$  and hence  $S$  is unsatisfiable.

**Theorem 9 (Completeness)** *Let  $S$  be a finite set of clauses. If  $S$  is unsatisfiable then  $\square \in Res^*(S)$ .*

**Proof:** Induction over the number  $n$  of atomic formulae in  $S$ .

With  $n = 0$  we have  $S = \{\square\}$  and hence  $\square \in S \subseteq Res^*(S)$ .

Assume  $n$  fixed and for every unsatisfiable set of clauses  $S$  with  $n$  atomic formulae  $A_1, \dots, A_n$  it holds that  $\square \in Res^*(S)$ .

Assume a set of clauses  $S$  with atomic formulae  $A_1, \dots, A_n, A_{n+1}$ . In the following we construct two clause sets  $S_f$  and  $S_t$ :

- $S_f$  is received from  $S$  by deleting every occurrence of  $A_{n+1}$  in a clause and by deleting every clause which contains an occurrence of  $\neg A_{n+1}$ .  
This transformation obviously corresponds to interpreting the atom  $A_{n+1}$  with *false*,
- $S_t$  results from a similar transformation, where occurrences of  $\neg A_{n+1}$  and clauses containing  $A_{n+1}$  are deleted, hence  $A_{n+1}$  is interpreted with *true*.

Let us show, that both  $S_f$  and  $S_t$  are unsatisfiable: Assume an assignment  $\mathcal{A}$  for the atomic formulae  $\{A_1, \dots, A_n\}$  which is a model for  $S_f$ . Hence the assignment

$$\mathcal{A}'(B) = \begin{cases} \mathcal{A}(B) & \text{if } B \in \{A_1, \dots, A_n\} \\ \text{false} & \text{if } B = A_{n+1} \end{cases} \text{ is a model for } S, \text{ which leads to a contradiction.}$$

A similar construction shows that  $S_t$  is unsatisfiable. Hence we can use the induction assumption to conclude that  $\square \in Res^*(S_t)$  and  $\square \in Res^*(S_f)$ .

Hence there is a sequence of clauses

$$C_1, \dots, C_m = \square$$

such that  $\forall 1 \leq i \leq m$  it holds  $C_i \in S_f$  or  $C_i$  is a resolvent of two clauses  $C_a$  and  $C_b$  with  $a, b < i$ . There is an analogous sequence for  $S_t$ :

$$C'_1, \dots, C'_t = \square$$

Now we are going to reintroduce the previously deleted literals  $A_{n+1}$  and  $\neg A_{n+1}$  in the two sequences:

- Clause  $C_i$  which has been the result of deleting  $A_{n+1}$  from the original clause in  $S$  are again modified to  $C_i \cup \{A_{n+1}\}$ . This results in a sequence

$$\overline{C_1}, \dots, \overline{C_m}$$

where  $\overline{C_m}$  is either  $\square$  or  $A_{n+1}$ .

- Analogous we introduce  $\neg A_{n+1}$  in the second sequence, such that  $\overline{C'_t}$  is either  $\square$  or  $\neg A_{n+1}$

In any of the above cases we get  $\square \in Res^*(S)$  latest after one resolution step with  $\overline{C_m}$  and  $\overline{C'_t}$ .

## Instantiated completeness proof

This is your set of current propositional clauses:

---

Currently empty.

---

### Current clauses manipulation:

Enter some Clauses:

Help:

[Delete selected](#) / [Delete all](#) current Clauses

[Add to](#) / [Overwrite](#) current clauses

[Convert selected](#) / [Convert all](#) current propositional formulas to CNF, replacing the current clauses.

### Save/Reload current clauses:

[Save](#) to database (enter name):

[Reload](#) from database (select name):

[Reload](#) from file (enter file name):

---

[Generate the resolution completeness of for the current clauses](#)  [Delete current proof](#)

This is your set  $S$  of current clauses :  $\{\}$

Unfortunately  $S$  is not unsatisfiable, so it does not make sense to explain the proof with it. Please try again with an unsatisfiable set of clauses.

Based on the theorems for correctness and completeness, we give a procedure for deciding the satisfiability of propositional formulae.

## Deciding Satisfiability of Propositional Formulae

Given a propositional formula  $F$ .

- Transform  $F$  into an equivalent CNF  $S$ .
- Compute  $Res^n(S)$  for  $n = 0, 1, 2, \dots$ 
  - If  $\square \in Res^n(S)$  then **Stop: unsatisfiable**.
  - if  $Res^n(S) = Res^{n+1}(S)$  then **Stop: satisfiable**.

**Theorem 10** *If  $S$  is a finite set of clauses, then there exists a  $k \geq 0$  such that*

$$Res^k(S) = Res^{k+1}(S)$$

Until now, we have been dealing with sets of clauses. In the following it will turn out, that it is helpful to talk about sequences of applications of the resolution rule.

**Definition 12** *A deduction of a clause  $C$  from a set of clauses  $S$  is a sequence  $C_1, \dots, C_n$ , such that*

- $C_n = C$  and
- $\forall 1 \leq i \leq n : (C_i \in S \text{ or } \exists l, r < i : C_i \text{ is a resolvent of } C_l \text{ and } C_r)$

*A deduction of the empty clause  $\square$  from  $S$  is called a refutation of  $S$ .*

### Example

We want to show, that the formula  $K = ((B \wedge \neg A) \vee C)$  is a logical consequence of  $F = ((A \vee (B \vee C)) \wedge (C \vee \neg A))$ . For this negate  $K$  and prove the unsatisfiability of  $F \wedge \neg K$

For this you can use the interaction in this book in various forms:

- Use the interaction *Truth Tables* for proving the unsatisfiability, or
- use the interaction *CNF Transformation* for transforming the formula into CNF, and then
- use the following interaction *Resolution*.

## Current Propositional Clauses

This is your set of current propositional clauses:

---

Currently empty.

---

Current clauses manipulation:	
Enter some Clauses: <input style="width: 90%;" type="text"/>  Help: <input style="width: 90%;" type="text"/>  <u>Add to</u> / <u>Overwrite</u> current clauses	<u>Delete selected</u> / <u>Delete all</u> current Clauses  <u>Convert selected</u> / <u>Convert all</u> current propositional formulas to CNF, replacing the current clauses.  <b>Save/Reload current clauses:</b> <u>Save</u> to database (enter name): <input style="width: 90%;" type="text"/> <u>Reload</u> from database (select name): <input style="width: 90%;" type="text"/> <u>Reload</u> from file (enter file name): <input style="width: 90%;" type="text"/>

---



### Interaction: Otter

Apply Otter to the current clauses   Delete current Otter Result

This is the result, (possibly from a previous run):

Currently none. □

### Interaction: Resolution Closure

Apply the “resolution closure” operator to the current set of clauses

Resulting set  $Res^*(\text{current clauses})$  (possibly from a previous run):

Currently empty.

□

Delete current resolution closure

#### Problem 1

Compute  $Res^n(M)$  for all  $n \geq 0$  and  $Res^*(M)$  for the following set of clauses:

1.  $M = \{\{A\}, \{B\}, \{\neg A, C\}, \{B, \neg C, \neg D\}, \{\neg C, D\}, \{\neg D\}$
2.  $M = \{\{A, \neg B\}, \{A, B\}, \{\neg A\}\}$
3.  $M = \{\{A, B, C\}, \{\neg B, \neg C\}, \{\neg A, C\}\}$
4.  $M = \{\{\neg A, \neg B\}, \{B, C\}, \{\neg C, A\}\}$

Which formula is satisfiable or which is unsatisfiable?

#### Problem 2

Indicate all resolvent of the clauses in S, where

$$S = \{\{A, \neg B, C\}, \{A, B, E\}, \{\neg A, C, \neg D\}, \{A, \neg E\}\}$$

#### Problem 3

Prove: A resolvent  $R$  of two clauses  $C_1$  and  $C_2$  is a logical consequence from  $C_1$  and  $C_2$ .

**Note:** Use the definition of "consequence".

**Problem 4**

Let  $M$  be a set of formulae and  $F$  a formula. Prove:

$$M \models F \text{ iff } M \cup \{\neg F\} \text{ is unsatisfiable.}$$

**Problem 5**

Compute  $Res^n(S)$  with  $n = 0, 1, 2$  and

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

**Problem 6**

Show that the following set  $S$  of formulae is unsatisfiable, by giving a refutation.  $S = (B \vee C \vee D) \wedge (\neg C) \wedge (\neg B \vee C) \wedge (B \vee \neg D)$

**Problem 7**

Show by using the resolution rule, that  $\neg A \wedge \neg B \wedge C$  is an inference from the set of clauses  $F = \{\{A, C\}, \{\neg B, \neg C\}, \{\neg A\}\}$ .

**Problem 8**

Show by using the resolution rule, that  $((P \rightarrow Q) \wedge P) \rightarrow Q$  is a tautology.

**Analytic Tableaux**

In this section we present a calculus, namely analytic tableau, which is an alternative to resolution. Although this calculus was developed independently from resolution, it will turn out that there are some interesting common features. The most obvious difference is, that analytic tableau work direct on formulae, there is no need to transform a formula in a clausal normal form.

**Tableaux - A Short History**

The development of tableaux calculi started in the 1950th. The first authors to be mentioned are Beth (1955), Hintikka (1955) and Schütte (1956). Their goal was primarily to develop calculi without meta-language constructs. Later in the 1960th the idea of derivation trees and nodes in such a tree labeled by formulae became famous as the concept of *analytic tableaux* introduced by Smullyan 1968. The idea of mechanisation of tableaux calculi was then introduced by Kanger 1957, Prawitz 1960, Wang 1960, Davis 1960 and Maslov 1968.

Later on the concept of analytic tableau was modified and refined for its use in automated deduction by Loveland 1968 (Model elimination), Kowalski, Kuehner 1971 (SL-resolution) and Bibel 1975, Andrews 1976 (Connection or matings methods). Nowadays there are numerous high performance theorem provers based on this work.

**The Calculus**

One of the advantages of tableaux calculi is that can be defined without transforming the formula into clause normal form.

**Example** Given a set of formulae  $\{\neg P \wedge \neg(Q \vee R), \neg(Q \wedge \neg R)\}$ . We are aiming at constructing a tree, whose branches contain nodes which are labeled with formulae.

For this it is important to analyse the formula according to its leading connective. Smullyan observed that some work can be saved if non-literal formulas are grouped into types which are treated identically:  $\alpha$  for formulas of conjunctive type,  $\beta$  for formulas of disjunctive type in the propositional case. Note that in the above example  $\neg(Q \wedge \neg R)$  has

to be treated as a formula of disjunctive type, because of the negation standing in front of the conjunction.

Correspondence between formulas and their types is summarised in Table 1.

$\alpha$	$\alpha_1, \dots, \alpha_n$	$\beta$	$\beta_1, \dots, \beta_n$
$\phi_1 \wedge \dots \wedge \phi_n$	$\phi_1, \dots, \phi_n$	$\phi_1 \vee \dots \vee \phi_n$	$\phi_1, \dots, \phi_n$
$\neg(\phi_1 \vee \dots \vee \phi_n)$	$\neg\phi_1, \dots, \neg\phi_n$	$\neg(\phi_1 \wedge \dots \wedge \phi_n)$	$\neg\phi_1, \dots, \neg\phi_n$
$\neg\neg\phi$	$\phi$		
$\neg\text{false}$	$\text{true}$		
$\neg\text{true}$	$\text{false}$		

Table 1: Correspondence of formulas and their types.

The letters  $\alpha$  and  $\beta$  are used to denote formulas of (and only of) the appropriate type. Let us now define the basic data structure for tableaux calculi together with the corresponding extension rules.

**Definition 13** A tableau for a logic  $L$  is a finitely branching tree whose nodes are formulas from  $L$ . A branch *branch* in a tableau  $T$  is a maximal path in  $T$ . When no confusion can arise, branches are frequently identified with the set of their nodes (formulas). Given a set  $\Phi$  of formulae from  $L$ , a tableau for  $\Phi$  is constructed by a (possibly infinite) sequence of applications of the following rules:

1. The tree consisting of a single node *true* is a tableau for  $\Phi$  (initialisation rule).
2. Let  $T$  be a tableau for  $\Phi$ ,  $B$  a branch of  $T$ , and  $\psi$  a formula in  $B \cup \Phi$ . If the tree  $T'$  is constructed by extending  $B$  by as many new linear subtrees as an instance of a tableau rule schema in Table 2 with premise  $\psi$  has extensions, and the nodes of the new subtrees are the formulas in the extensions of the rule instance, then  $T'$  is a tableau for  $\Phi$  (expansion rule).

$\frac{\alpha}{\alpha_1}$	$\frac{\beta}{\beta_1 \mid \dots \mid \beta_n}$
$\vdots$	
$\alpha_n$	

Table 2: Rule schemata for tableaux.

One tableau for our example is depicted in figure 2.

**Definition 14** In a tableau  $T$  for a set  $\Phi$  of sentences a branch  $B$  is closed iff  $B \cup \Phi$  contains a pair  $\phi, \neg\phi$  of complementary formulas, or *false*; otherwise, it is open. A tableau is closed if all its branches are closed.

A tableau proof for (the unsatisfiability of) a set  $\Phi$  of formulae is a closed tableau  $T$  for  $\Phi$ .

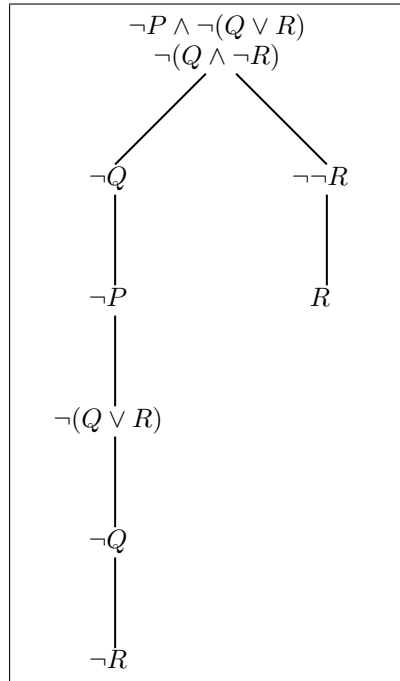


Figure 2: A tableau for a formula in propositional logic

### Interaction: Tableaux

This is our current set of formulas:

---

Currently empty.

---

#### Current formulas manipulation:

[Add](#) random formulas of complexity  
                   low        medium        high

Enter some formulas:                      [Help](#):

[Add to](#) / [Overwrite](#) current formulas

[Delete selected](#) / [Delete all](#) current formulas

#### Save/Reload current formulas:

[Save](#) to database (enter name):

[Reload](#) from database (select name):

[Reload](#) from file (enter file name):

---

### [Interactive Tableaux for current propositional formulas](#)

[Construct a tableau](#) for current propositional formulas.

Note (system limitations): (1) only trees with a branching factor not exceeding 5 can be drawn. (2) the time limit for tableau construction is 60 sec.



And here is the resulting tableau (possibly from a previous run):

Currently none. [Delete current tableau](#)

**Problem 1**

Give a strict tableau proof for the following formulae:

1.  $\neg((A \downarrow B) \downarrow (A \vee B))$
2.  $((A \rightarrow B) \wedge (B \rightarrow C)) \rightarrow (\neg(\neg C \wedge A))$

**Clause Normalform Tableau**

Let us now refine the calculus for the special case, that we deal with sets of clauses, which represent a formula in CNF. Note that in the case of conjunktive normal form clauses we only have literals, which are connected by  $\vee$ -junctors. Hence every clause is of  $\beta$ -type. In figure a tableau for a set of clauses is given. The clauses from the given clause set form the initial tableau; then there is only the  $\beta$ -rule applicable for further extensions of the tableau.

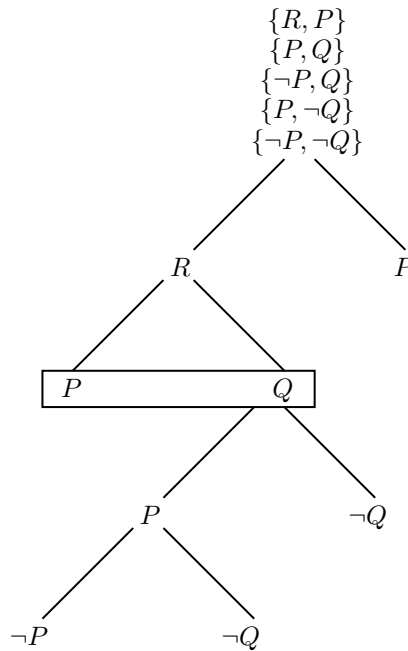


Figure 3: A tableau for clauses in CNF

In the following formal definition of a clause normal form tableau we start with an initial tableau, which is formed by taking an arbitrary clause from the given clause set  $S$ . For further extensions of the tableau  $\beta$ -rule-applications with clauses from  $S$  can be used. Which of the clauses is allowed is controlled by a *link condition*.

**Definition 15** A clause (normalform) tableau for a set of clauses  $S$  is a tableau for  $S$ , whose nodes are literals from  $S$  and which is constructed by a (possibly infinite) sequence of applications of the following rules:

1. The tree consisting of root true and immediate successors  $L_1, \dots, L_n$ , where  $C = L_1, \dots, L_n \in S$  is a tableau for  $S$  (initialisation rule).
2. Let  $T$  be a tableau for  $S$ ,  $B$  a branch of  $T$ , and  $C = L_1, \dots, L_n \in S$ , such that the link-condition (see below) is satisfied. If the tree  $T'$  is constructed by extending  $B$  by the  $n$  subtrees  $L_i$ , then  $T'$  is a tableau for  $S$  (expansion rule).

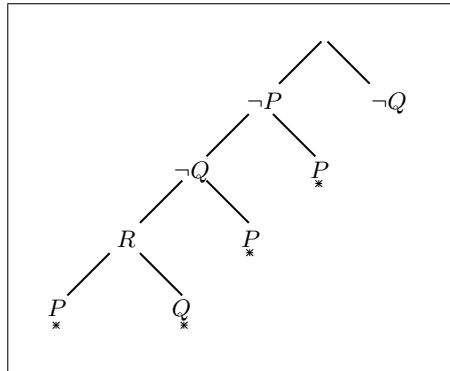
The following are three possible link conditions:

1. No condition.
2. Weak link condition: There is a literal  $L \in B$  and  $\bar{L} \in C$ .
3. Strong link condition: Let  $L$  be the leaf of  $B$ , then there is  $\bar{L} \in C$ .

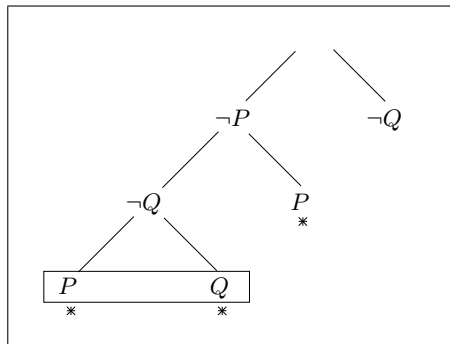
In fact we have defined three different calculi:

- Without link condition it is called clause normal form tableau,
- with the weak link condition we call it connection calculus, and
- with the strong link condition it is called model elimination.

An example for clause normal form tableau was given in figure ). An example for a connection calculus tableau is



and finally a model elimination tableau is given by



**Theorem 11** Clause normal form tableau are complete.

Note that strong link condition do not allow for confluent <sup>1</sup> proof procedures. If no link condition (i.e. the empty one) is applied it is trivial to get a confluent version. For the case of weak link condition this is not obvious.

In order to arrive at a decision procedure for propositional clauses we need an extra condition:

**Definition 16** *A branch  $B$  of a tableau for a clause set  $S$  is called regular, if no literal occurs more than once.*

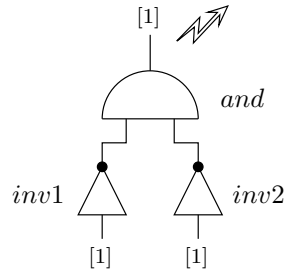
**Theorem 12** *Clause normal form tableau with regularity and link condition give a decision procedure for propositional logic.*

---

<sup>1</sup> Let  $M$  be a set and  $\rightarrow$  a binary relation on  $M$ . Then  $(M, \rightarrow)$  is called *confluent* if

for all  $u, v, x, y$  and  $u \rightarrow^* x$  and  $u \rightarrow^* y$  there is a  $z$  with  $x \rightarrow^* z$  and  $y \rightarrow^* z$

**Problem (diagnosis).** Consider this electronic circuit with two input lines and one output line:



Suppose, as depicted, that both input lines are “1” and that the output line is “1”, thus contradicting the expected output value “0”.

1. First, formalize the circuit, i.e. the functionality of the three components and the two connections by neglecting the possibility of not correctly functioning components (i.e. do not use *abnormal*-Literals).
2. Consider the value pairs “0-0”, “1-0” and “1-1” to be supplied to the input lines. For each of these, using the result from (1) compute the expected output value by means of analytic tableau.

How did you read off the results from the tableaux?

3. Use your formalization and analytic tableau to prove that the output value “1” contradicts the expected behavior in case of input “1-1”.

How did you read off the result from the tableau?

4. Now modify the formalization of the components in (1) by *abnormal*-literals as shown in class.

Use analytic tableau to compute all possible diagnosis for the input “1-1” and output “1”.

How did you read off the result from the tableau?

## Layout

[Format document for screen](#)