

Notes for the Logic Course 2010/2011

Propositional and First Order Logic

Rosella Gennari

<http://www.inf.unibz.it/~gennari>
gennari@inf.unibz.it

Computer Science

Free University of Bozen-Bolzano
January 16, 2011

Disclaimer. The course notes are meant as complementary material for the students of the course of Logic at the Free University of Bozen-Bolzano. They are by no means exhaustive and students need refer to the course textbook whenever indicated. The notes and exercises are periodically updated and made available in the course web page: <http://www.inf.unibz.it/~gennari/index.php?page=logic>.

Contents

1	Propositional Logic	3
1.1	Language	3
1.1.1	Propositional Symbols	3
1.1.2	Propositional Formulae	4
1.1.3	Parsing Tree	5
1.1.4	Unique Readability	7
1.1.5	Induction and Recursion	10
1.2	Semantics	11
1.2.1	Interpretations	11
1.2.2	Satisfiability and Validity	14
1.2.3	Satisfiability of a Set of Formulae	15
1.2.4	Entailment	16
1.3	Proof Theory	19
1.3.1	Proof Calculi	19
1.3.2	An Analytic Tableau Calculus	21
1.3.3	The DPLL Calculus (Optional for the Course)	32
1.4	Summary	35
1.4.1	Language	35
1.4.2	Semantics	35
1.4.3	Proof Theory	36
2	First Order Logic	37
2.1	Language	37
2.1.1	Lexicon and Expressions	38

2.1.2	Terms and Formulae	39
2.1.3	Closed Formulae	43
2.1.4	Examples of First Order Formalisation	44
2.2	Semantics	47
2.2.1	Interpretations	47
2.2.2	Substitutions	48
2.2.3	Satisfiable and Valid Formulae	50
2.2.4	Set Satisfiability	53
2.2.5	Entailment and Logical Equivalence	56
2.3	Proof Theory	58
2.3.1	Analytic Tableau Rules	58
2.3.2	Tableaux	59
2.3.3	Satisfiability and Validity	62
2.3.4	Set Satisfiability and Entailment	65
2.4	Summary	68
2.4.1	Language	69
2.4.2	Semantics	69
2.4.3	Proof Theory	69

Chapter 1

Propositional Logic

1.1 Language

There is one special type of construction that occurs in logic and in other areas like number theory. We construct a certain set by starting with some initial atomic elements, and applying certain operations to them over and over again.

Its members will *only* be the initial elements, and what can be built up from the initial elements by applying the operations some finite number of times.

1.1.1 Propositional Symbols

So, what are such atomic elements we should start with in order to build a propositional language? They are the **propositional symbols**, denoted by $p, q, p_0, \dots, p_n, \dots$. We assume to have countably many of them.

We usually use P to denote the set of propositional symbols, that is, the **signature** of the propositional language.

The operations are given by the **logical symbols** (a.k.a., **connectives**):

- (1) $\perp$ and $\top$, that are 0-ary;
- (2) $\neg$, that is unary;
- (3) $\wedge, \vee, \rightarrow, \leftrightarrow$, that are binary.

Atoms of $\text{PL}(P)$ are either propositional symbols (e.g., p), $\perp$ or $\top$. No other expression is an atom.

In order to define a propositional language, we also need **auxiliary symbols** (a.k.a., **parentheses**): $(,)$.

The propositional **lexicon** or **alphabet** over P is the set of all the above symbols, namely: atoms, connectives, parentheses.

1.1.2 Propositional Formulae

Then, how do we *combine* the symbols of a lexicon so as to obtain the formulae of a propositional language? First by juxtaposition, creating the so-called expressions or words of the language.

A **propositional expression** or **word** over P is any finite sequence $s_1 \dots s_n$ of symbols of the lexicon over P .

Example 1. Take $P = \{p\}$. $p)\neg p$ is an expression over P . $\neg q$ is not an expression over P .

$\text{Expr}(P)$ denotes the set of expressions over P . A **language** over P is any subset of $\text{Expr}(P)$.

Now, what distinguishes the language of propositional formulae from the other languages? The fact that a formula of $\text{Expr}(P)$ is an expression of $\text{Expr}(P)$ with a specific grammatical structure.

The definition of formula in this subsection diverges from the one based on parsing tree in Section 3.2 of [Chiswell and Hodges, 2007], albeit the two definitions are equivalent; we will prove it later. Hereby, we give the standard *inductive and recursive* definition of formulae as specific expressions, that is, as specific flat sequences of propositional symbols.

Definition 1. Let ϕ and ϕ' represent expressions of $\text{Expr}(P)$:

- (B) if $p \in P \cup \{\perp, \top\}$ then p is a propositional formula over P ;
- (I)
 - if ϕ is a propositional formula over P then $(\neg\phi)$ is a propositional formula over P ;
 - if ϕ and ϕ' are propositional formulae over P then $(\phi \square \phi')$ are propositional formulae over P , for all $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$;
- (S) no other expression of $\text{Expr}(P)$ is a propositional formula over P .

The **propositional language** over P is the set consisting precisely of the propositional formulae over P . We denote it by $\text{PL}(P)$.

Note 1. The above inductive (and recursive) definition of propositional formulae represents them as *flat* sequences. There are several equivalent definitions of formulae as sequences. One is, mostly, a rewriting: instead of extensively writing (B), (I), and (S), one often gives them in Backus Naur Form (BNF):

$$\phi, \psi \longrightarrow p \mid \perp \mid \top \mid (\neg\phi) \mid (\phi \wedge \psi) \mid (\phi \vee \psi) \mid (\phi \rightarrow \psi) \mid (\phi \leftrightarrow \psi)$$

More subtly, clause (S) is equivalent to saying that

(S') $\text{PL}(P)$ is the *smallest* subset of expressions over P that satisfy the closure properties (B) and (I).

We will use such a characterisation of formulae in proofs concerning properties of formulae later on. This and other equivalent definitions are discussed in [Mendelson, 1997].

Notice also that our definition of propositional formulae is analogous to that of natural numbers starting, for instance, from real numbers:

- replace (I) with “0 is a natural number”;
- replace (B) with “if n is a natural number then $n + 1$ (the successor of n) is a natural number”;
- replace (S) with “no other real number is a natural number”.

We can prove properties $S(n)$ about natural numbers, like

$$\sum_{i=1}^n i = \frac{n \times (n + 1)}{2}, \tag{1.1}$$

by **induction**, using their definition. Similarly, we can prove statements about propositional formulae by appealing to their inductive definition above. We will see how next.

1.1.3 Parsing Tree

What is a parsing tree for $\phi \in \text{PL}(P)$? First of all, it is a binary tree (N, D) , see [Chiswell and Hodges, 2007]). Then it has a right labelling specified as follows.

Definition 2. Let (N, D) be a binary tree, and r a function from the set of nodes to the lexicon of $\text{PL}(P)$. We call (N, D, r) a parsing tree for $\text{PL}(P)$, and r its right labelling, if:

- for every leaf **node**, $r(\text{node})$ is an atom of $\text{PL}(P)$;
- for every **node** of arity 1, $r(\text{node}) = \neg$;
- for every **node** of arity 2, $r(\text{node})$ is equal to a binary connective.

In other words, a parsing tree is a binary tree with lexicon symbols of $\text{Expr}(P)$ as right labels. Next, we can left label parsing trees for $\text{PL}(P)$ with the *parsing left labelling* $\delta(\text{node})$ as in Figure 1.1.

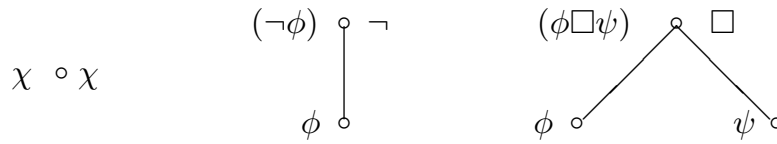


Figure 1.1: Parsing left labelling.

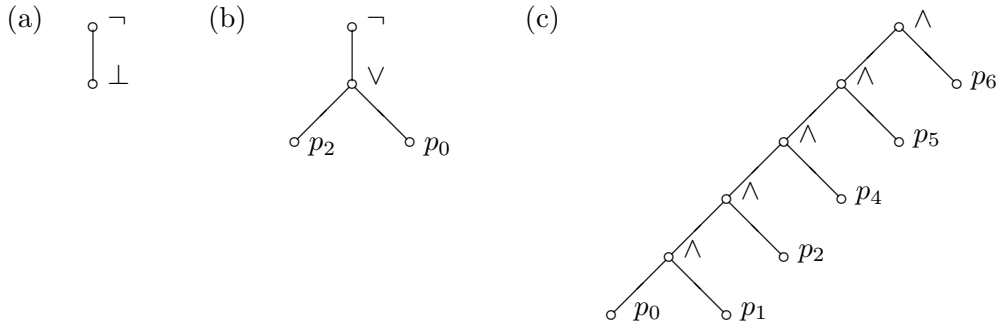


Figure 1.2: Parsing trees.

Note 2. The parsing left labelling of a parsing tree is uniquely determined by the right labelling and the above clauses—prove it by induction on the height of the tree! This justifies the fact that we refer to δ as *the* parsing left labelling.

Clearly, every $\delta(\text{node})$ of a parsing tree is a formula.

Lemma 1. Let (N, D, r) be a parsing tree with parsing left label δ . For each **node** of N , $\delta(\text{node})$ is a propositional formula of $\text{PL}(P)$.

Proof. By induction on the height of **node**. □

The formula $\delta(\text{root})$ is the formula **associated** to (or **generated** by) the parsing tree.

Example 2 (Generation). The formulas associated to the parsing trees of Figure 1.2 are, respectively: (a) $(\neg \perp)$, (b) $(\neg(p_2 \vee p_0))$, (c) $((p_0 \wedge p_1) \wedge p_2) \wedge \dots p_6$.

Vice versa, every formula of $\text{PL}(P)$ is associated to a parsing tree.

Lemma 2. Every $\phi \in \text{PL}(P)$ is associated to a parsing tree.

Proof. We prove the statement by induction on ϕ .

(B) If ϕ is an atom, then we create a tree with a single node and no descendants. The left and right labels of the node are equal to ϕ .

(I) If ϕ is not an atom, then ϕ is of the form $(\neg\psi)$ or $(\psi \square \psi')$ with $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.

1. Let us consider the case $\phi = (\neg\psi)$. Assume (induction hypothesis) that ψ is associated to a parsing tree (N, D, r) with root m . Let $n \notin N$ and create the tree $T = (N \cup \{n\}, D \cup \{(n, m)\})$. Right label n with $\neg$ and left label it with $(\neg\psi)$. Since (N, D, r) is a parsing tree, so is T and is associated to $(\neg\psi)$.
2. Let us consider the case $\phi = (\psi \wedge \psi')$. Assume (induction hypothesis) that ψ is associated to the parsing tree (N, D, r) with root m , and ψ' is associated to the parsing tree (N', D', r') with root m' ; we can safely assume that $N \cap N' = \emptyset$. Let $n \notin N \cup N'$ and create the tree $T = (N \cup N' \cup \{n\}, D \cup D' \cup \{(n, m)\})$. Right label n with $\wedge$ and left label it with $(\psi \wedge \psi')$. Since both (N, D, r) and (N', D', r') are parsing trees, so is T and is associated to $(\psi \wedge \psi')$. The other binary connectives are dealt with similarly.

Since no other expression is a formula, the proof is concluded. $\square$

The proof of the above lemma gives us a recipe for constructing the parsing tree of a formula. However, it does not tell us how to detect the so-called main connective of a formula, e.g., $\wedge$ in $(\phi \wedge \psi)$. We will see how to do it in the next section.

The above two lemmas together allow us to conclude the following.

Theorem 1. $\phi \in \text{PL}(P)$ iff it is associated to a parsing tree.

1.1.4 Unique Readability

By definition, every formulae ϕ of $\text{PL}(P)$ can be written in one of the forms of Definition 1, and has a parsing tree. Is the writing of the formula unique? In other words, is the associated parsing tree unique?

The answer is yes. We will not prove it completely, but only give the main ideas of the proof. In order to prove it we show that the so-called main connective of the formula is uniquely defined. For proving this, we will count the number of non-closed parentheses of an expression in a formula, as in [Chiswell and Hodges, 2007]. So, let $s_1 \dots s_i \dots s_n$ be a propositional expression.

- If l is the number of left parentheses of the expression and r that of right parentheses, then $l - r$ is the **parenthesis depth of the expression**.
- If s is a symbol of the lexicon, every s_i equal to s is an **occurrence** of s in the expression. The **parenthesis depth of the occurrence** s_i in the expression is the parenthesis depth of the initial segment $s_1 \dots s_i$ of the expression.

Example 3. Consider $(p \wedge (\neg q))$. The parenthesis depth of $(p \wedge (\neg q))$ is 0. Then p has one occurrence in the expression. The parenthesis depth of p in the expression is 1.

Consider $p \wedge ((\neg p) \vee p)$. Then p has three occurrence in the expression. The parenthesis depth of the left-most occurrence of p in the expression is 0; that of the middle occurrence is 2, equal to that of $p \wedge ((\neg p)$; that of the rightmost occurrence is 1, equal to that of $p \wedge ((\neg p) \vee p$.

Next, we show that a formula can be written in a unique form, and hence is associated to a unique parsing tree. In particular, the following theorem states that a non atomic formula, that is, a **complex** formula, has a unique occurrence of a connective with parenthesis depth equal to 1.

Theorem 2 (Unique Readability). Any propositional formula of $PL(P)$ has *exactly one* of the following forms:

- i. an atom;
- ii. $(\neg \phi)$ where ϕ is a $PL(P)$ formula; no other occurrence of connectives besides $\neg$ in $(\neg \phi)$ has parenthesis depth 1;
- iii. $(\phi \square \phi')$ where $\square$ is any binary connective, ϕ and ϕ' are $PL(P)$ formulae; no other occurrence of connectives besides the one shown of $\square$ in $(\phi \square \phi')$ has parenthesis depth 1.

Proof. Any formula has such a form by definition. Uniqueness can be proved by induction on ϕ , see the proof schema in [Chiswell and Hodges, 2007]. $\square$

The occurrence of the connective with parenthesis depth equal to 1 is the unique **main connective** of the formula.

Hereby, we build the (now we know) unique parsing tree of a formula, following the recipe in the proof of Lemma 2, and detect the main connective by computing its parenthesis depth.

Example 4. Consider $(q \vee (p \rightarrow p))$. Define **root** with left label equal to the formula. Scan the formula from the left. The first symbol is “(” so the expression is complex. The connective with parenthesis depth equal to 1 is $\vee$. This is the unique main connective according to the above theorem, and thus place it as right label of **root**. Now, add two children to the root: place q as left label of the left more child, and $(p \rightarrow p)$ as left label of the right more child. Restart the procedure from q . This is atomic, so the right label is equal to the left label. Restart the procedure with $(p \rightarrow p)$. The main connective is now $\rightarrow$. Place it as right label. Add two children: place the first occurrence of p as left label of the left more child, and the second occurrence of p as left label of the right more child. Restart from them and conclude the parsing tree (re)construction and recognise that $(q \vee (p \rightarrow p))$ is a propositional formula.

Since every formula is associated to a unique parsing tree, and parsing trees are only associated to formulas, we can use parsing trees to recognise whether an expression is or not a formula.

Example 5 (Recognition). Take $P = \{p, q\}$. Consider the following expressions over P :

1. $(p \vee q)$
2. (p)
3. $p \wedge q$
4. $(p \wedge (q \vee q))$

Only the first and the last expressions are proposition formulae, that is, they are associated to a parsing tree. Do construct these. $\square$

In particular, parsing trees allow us to give an operational definition of the subformulae of a formula.

Definition 3. Let ϕ a formula and T its parsing tree with parsing left labelling δ . For each **node** of T , $\delta(\text{node})$ is a **subformula** of ϕ .

Example 6. Consider $\phi = (p \rightarrow ((\neg p) \wedge q))$. The distinct subformulae of ϕ are: ϕ ; $((\neg p) \wedge q)$; $(\neg p)$; p ; q .

Note 3. Albeit parentheses allow for unique readability they make formulae difficult to read for humans. Therefore, we convey the following on omitting parentheses:

- outermost parentheses can be dropped; e.g., $p \rightarrow q$ stands for $(p \rightarrow q)$; $\neg p$ stands for $(\neg p)$;
- the following binding order is adopted: $\neg$ binds more than $\wedge$ and $\vee$, which in turn bind more than both $\leftrightarrow$ and $\rightarrow$; e.g., $\neg p \rightarrow q$ stands for $((\neg p) \rightarrow q)$; $p \wedge q \rightarrow \neg q$ stands for $((p \wedge q) \rightarrow (\neg q))$;
- where one connective is used repeatedly, parentheses can be dropped and grouping is to the right: $p \wedge q \wedge r$ stands for $(p \wedge (q \wedge r))$.

A formula so simplified can always be rewritten in its unique, grammatically correct form—surely, before recognising it with the above parsing algorithm...

We will adopt such conventions whenever they simplify readability for humans. $\square$

1.1.5 Induction and Recursion

How to prove properties $\mathcal{A}$ concerning $\text{PL}(P)$ formulas? We can prove such properties by directly using the inductive definition of formulas. The following theorem gives us the procedure.

Theorem 3 (Induction). Let $\mathcal{A}$ be a property of expressions over P . Then $\mathcal{A}$ holds for all propositional formulas over P if:

- i. it holds for any atom;
- ii. if it holds for ϕ then it holds for $(\neg\phi)$;
- iii. if it holds for ϕ and ϕ' , then it holds for $(\phi \square \phi')$, where $\square$ is any binary connective.

Proof. Let $X \subseteq \text{PL}(P)$ be the subset of formulas that satisfies $\mathcal{A}$. Atoms belong to it by (i). X is closed for the connectives by (ii) and (iii). Since $\text{PL}(P)$ is the smallest subset of expressions that contains the atoms and is closed for the connectives, then $\text{PL}(P) \subseteq X$. $\square$

For instance, the proof of Lemma 2 was done by induction over formulae.

Moreover, thanks to the Uniqueness Readability Theorem, we can *uniquely* recursively ‘extended’ functions with domain P to functions with domain the *entire* $\text{PL}(P)$.

Theorem 4 (Recursion). Let A be any set and f a function from the atoms of $\text{PL}(P)$ into A . Consider functions over A as follows: $H_\square : A \times A \mapsto A$, one for each $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$; $H_\neg : A \mapsto A$. Then the mapping F from the propositional formulae into A that satisfies the following constraints

- $F(\text{atom}) = f(\text{atom})$,
- $F(\neg\phi) = H_\neg(F(\phi))$,
- $F(\phi\square\psi) = H_\square(F(\phi), F(\psi))$,

is unique.

Example 7. Consider

1. $\text{height}(p) = \text{height}(\perp) = \text{height}(\top) = 0$.

Extend it to $\text{height}(\phi)$, for any $\phi \in \text{PL}(P)$, as follows:

2. $\text{height}(\neg\phi) = 1 + \text{height}(\phi) = 0$;
3. $\text{height}(\phi\square\phi') = 1 + \max\{\text{height}(\phi), \text{height}(\phi')\}$.

The uniqueness theorem guarantees such a definition is well done: any function with the same behaviour over the atoms and satisfying clause 2 and 3 will be equal to height .

1.2 Semantics

1.2.1 Interpretations

An **interpretation** (P -**structure** in [Chiswell and Hodges, 2007]) of the propositional symbols in P is a function $\mathcal{I}: P \mapsto \{\mathbf{T}, \mathbf{F}\}$.

Next, we can extend the definition of $\mathcal{J}$, uniquely (why?), over all formulas of $\text{PL}(P)$ as follows:

$$\begin{aligned}
 \mathcal{J}(\perp) &= \text{F}, \\
 \mathcal{J}(\top) &= \text{T}, \\
 \mathcal{J}(\neg\phi) &= \text{T} \quad \text{iff} \quad \mathcal{J}(\phi) = \text{F}, \\
 \mathcal{J}(\phi \wedge \phi') &= \text{T} \quad \text{iff} \quad \mathcal{J}(\phi) = \text{T} \text{ and } \mathcal{J}(\phi') = \text{T}, \\
 \mathcal{J}(\phi \vee \phi') &= \text{T} \quad \text{iff} \quad \mathcal{J}(\phi) = \text{T} \text{ or } \mathcal{J}(\phi') = \text{T}, \\
 \mathcal{J}(\phi \rightarrow \phi') &= \text{T} \quad \text{iff} \quad \mathcal{J}(\phi) = \text{F} \text{ or } \mathcal{J}(\phi') = \text{T}, \\
 \mathcal{J}(\phi \leftrightarrow \phi') &= \text{T} \quad \text{iff} \quad \mathcal{J}(\phi) = \mathcal{J}(\phi').
 \end{aligned}$$

If $\mathcal{J}(\phi) = \text{T}$ we write $\mathcal{J} \models \phi$, and say that $\mathcal{J}$ **satisfies** ϕ , or makes ϕ **true**, or is a **model** of ϕ .

Else (i.e., $\mathcal{J}(\phi) = \text{F}$) we write $\mathcal{J} \not\models \phi$, and say that $\mathcal{J}$ does not satisfy ϕ , makes ϕ false, or is not a model of ϕ .

Equivalently, in words:

$$\begin{aligned}
 \mathcal{J} &\not\models \perp, \\
 \mathcal{J} &\models \top, \\
 \mathcal{J} \models (\neg\phi) &\quad \text{iff} \quad \mathcal{J} \not\models \phi, \\
 \mathcal{J} \models (\phi \wedge \phi') &\quad \text{iff} \quad \mathcal{J} \models \phi \text{ and } \mathcal{J} \models \phi', \\
 \mathcal{J} \models (\phi \vee \phi') &\quad \text{iff} \quad \mathcal{J} \models \phi \text{ or } \mathcal{J} \models \phi', \\
 \mathcal{J} \models (\phi \rightarrow \phi') &\quad \text{iff} \quad \mathcal{J} \not\models \phi \text{ or } \mathcal{J} \models \phi', \\
 \mathcal{J} \models (\phi \leftrightarrow \phi') &\quad \text{iff} \quad \text{either } \mathcal{J} \models \phi \text{ and } \mathcal{J} \models \phi' \text{ or } \mathcal{J} \not\models \phi \text{ and } \mathcal{J} \not\models \phi'.
 \end{aligned}$$

Some observations are in order.

Note 4. First, $\mathcal{J} \models \phi \vee \phi'$ is for inclusive disjunction, as in “in order to pass the exam question 3 or question 4 must have been answered correctly”. Clearly, you will not be penalised if both answers are correct!

Second, $\mathcal{J} \models \phi \rightarrow \phi'$ is much debated. Historically, its interpretation comes from the way of reading “if...then” statement as in mathematics, and hence in the (standard) semantics of programming languages in computer science. However, such a truth-value semantics does not fully render the common sense meaning of the conditional “if-then” in natural language. E.g., does having a false antecedent (condition) ϕ really make the implication if- ϕ -then- ψ true, independently of the interpretation of the consequent ψ ?

Third, thanks to the Uniqueness Readability Theorem, $\mathcal{J}$ over P is uniquely determined by the values it assigns to the propositional symbols of P , and the above clauses. But what really matters for claiming whether $\mathcal{J} \models \phi$, or not, is the values that $\mathcal{J}$ assigns to the propositional symbols *occurring* in ϕ —and we can forget about all the others of P . This is the contents of the following proposition.

Lemma 3. If, for all distinct propositional symbols p occurring in ϕ , $\mathcal{I} \models p$ iff $\mathcal{I}' \models p$, then $\mathcal{I} \models \phi$ iff $\mathcal{I}' \models \phi$.

Proof. By induction on ϕ . □

This allows us to count the *different* interpretations of a formula.

Theorem 5. Let $\phi(p_1, \dots, p_n)$ be a formula with n distinct proposition symbols $p_1, \dots, p_n$. The number of different interpretations of ϕ is 2^n .

Proof. The interpretations of $p_1, \dots, p_n$ are mappings from $\{p_1, \dots, p_n\}$ to $\{\mathbf{T}, \mathbf{F}\}$. Their number is 2^n . Any interpretation of ϕ is uniquely determined by the interpretation of its n distinct propositional symbols by Lemma 3. Therefore the number of different interpretations of ϕ is 2^n . □

Consider the 2^n *different* interpretations for the symbols $p_1, \dots, p_n$. Their values can be listed in a table of the following form, with 2^n rows.

p_1	p_2	$\dots$	p_n
$\mathcal{I}_1(p_1)$	$\mathcal{I}_1(p_2)$	$\dots$	$\mathcal{I}_1(p_n)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathcal{I}_{2^n}(p_1)$	$\mathcal{I}_{2^n}(p_2)$	$\dots$	$\mathcal{I}_{2^n}(p_n)$

The table can be extended to compute the value of each interpretation (one per row) on the complex formula $\phi(p_1, \dots, p_n)$. We will give examples of how to do it. See also Section 3.5 of [Chiswell and Hodges, 2007].

Example 8. Let $\phi = (p \rightarrow (\neg q))$. Its truth table is given below.

p	q	$(p \rightarrow (\neg q))$			
1	1	1	1	1	0
1	0	1	0	0	1
0	1	0	1	1	0
0	0	0	1	0	1

Let $\phi = (p \vee (\neg p))$. Its truth table is given below.

p	$(p \vee (\neg p))$			
1	1	1	0	1
1	1	1	0	1
0	0	1	1	0
0	0	1	1	0

1.2.2 Satisfiability and Validity

A formula ϕ is

- **satisfiable** (a.k.a., **consistent**) if there exists $\mathcal{J}$ that satisfies ϕ ,
- **unsatisfiable** (a.k.a., **inconsistent**) otherwise (i.e., for all $\mathcal{J}$, $\mathcal{J}$ does not satisfy ϕ),

and

- **valid** (a.k.a., a **tautology**) if all $\mathcal{J}$ satisfy ϕ ,
- **falsifiable** (a.k.a., **contingent**) otherwise (i.e., there exists $\mathcal{J}$ that does not satisfy ϕ).

Truth tables allow us to decide whether a formula is satisfiable or not, valid or not.

Theorem 6 (Decidability 1).

There is an algorithm which, given any ϕ in $\text{PL}(P)$, decides whether the formula is satisfiable or not.

There is an algorithm which, given any ϕ in $\text{PL}(P)$, decides whether the formula is valid or not.

Proof. Let ϕ be a formula with n distinct propositional symbols. By the definition above, we have the following, for any ϕ of $\text{PL}(P)$:

- if all the cells under its main connective are filled in with **T** then ϕ is valid,
- else (at least one cell under its main connective is filled in with **F**), ϕ is falsifiable,

and

- if all the cells under its main connective are filled in with **F** then ϕ is unsatisfiable,
- else (at least one cell under its main connective is filled in with **T**), ϕ is satisfiable.

We thus build the truth table of ϕ with 2^n rows (see Theorem 5) and check which of the above conditions is true in 2^n steps in the worst-case. $\square$

Example 9. Let us consider the formulae of Example 8. The first is satisfiable and falsifiable. The second is a tautology. Its negation is unsatisfiable—do build its truth table and check that the negated formula is unsatisfiable.

The above proof suggests that deciding on the satisfiability (falsifiability) of ϕ requires *choosing* (guessing) and building one row only of the truth table *at a time*: if T (F) is under the main connective of ϕ , we can stop building the table and return that the formula is satisfiable (falsifiable).

Notice also that, having negation in the language of propositional logic, satisfiability and validity are complementary problems: a formula is valid iff its negation is unsatisfiable.

Theorem 7. For every propositional formula ϕ , we have the following equivalences: ϕ is valid iff $\neg\phi$ is unsatisfiable.

Proof. Consider any interpretation $\mathcal{I}$: now, $\mathcal{I} \models \neg\phi$ iff, by definition, $\mathcal{I} \not\models \phi$ (*). By definition, $\neg\phi$ is a tautology iff, *for all interpretations* $\mathcal{I}$, $\mathcal{I} \models \neg\phi$. By (*), this is equivalent to saying that, *for all interpretations* $\mathcal{I}$, $\mathcal{I} \not\models \phi$. By definition, this means that ϕ is unsatisfiable. $\square$

Thanks to the above, a decision procedure for the satisfiability of $\neg\phi$ can be used for the validity of ϕ —the computational costs can be different, however.

1.2.3 Satisfiability of a Set of Formulae

The notion of model can be extended to sets of formulae Θ as follows:

$$\mathcal{I} \models \Theta \quad \text{iff} \quad \mathcal{I} \models \phi \quad \text{for all } \phi \in \Theta.$$

We will say that Θ is **satisfiable** (a.k.a., **consistent**) if it has a model. It is **unsatisfiable** (a.k.a., **inconsistent**) if it has no model.

Example 10. The set $\Theta = \{p, \neg p\}$ is unsatisfiable: no model of p is also a model of $\neg p$.

Truth tables can be used to decide on whether a finite Θ is satisfiable or not, by treating Θ as the finite conjunction of its formulas, as stated below.

Theorem 8 (Decidability 2). Let $\Theta = \{\phi_1, \dots, \phi_n\}$ a finite set of $\text{PL}(P)$ formulas.

- For every interpretation $\mathcal{I}$ for $\text{PL}(P)$, $\mathcal{I} \models \Theta$ iff $\mathcal{I} \models \bigwedge_{i=1}^n \phi_i$.

- There is an algorithm which, given any finite set of formulae Θ , will decide whether Θ is satisfiable or not.

Proof. Now, $\mathcal{J} \models \bigwedge_{i=1}^m \phi_i$ iff $\mathcal{J} \models \Theta$ by the above definition. Since the truth table method decides on $\bigwedge_{i=1}^m \phi_i$ by Theorem 6, it also decides on the satisfiability of Θ . $\square$

Example 11. The set $\Theta = \{p, \neg p\}$ is unsatisfiable. We can decide on it by building the following truth table.

$(p$	$\wedge$	$(\neg$	$p))$
T	F	F	T
F	F	T	F

1.2.4 Entailment

Next, we define the notion of entailment between sets of formulas and a formula. We say that Θ **entails** ϕ , and write $\Theta \models \phi$, if ϕ is true in all models of Θ , that is,

$$\text{for all } \mathcal{J}, \text{ if } \mathcal{J} \models \Theta \text{ then } \mathcal{J} \models \phi.$$

Having $\neg$ in the language, entailment and set (un)satisfiability are complementary problems in the following sense.

Lemma 4. $\Theta \models \phi$ iff $\Theta \cup \{(\neg\phi)\}$ is unsatisfiable.

Proof. Let us spell out what “ $\Theta \models \phi$ ” and “ $\Theta \cup \{(\neg\phi)\}$ ” mean.

(1) $\Theta \models \phi$ means the following: if $\mathcal{J} \models \Theta$ then $\mathcal{J} \models \phi$.

(2) $\Theta \cup \{(\neg\phi)\}$ is unsatisfiable means the following: if $\mathcal{J} \models \Theta$ then $\mathcal{J} \not\models \neg\phi$.

Therefore proving that (1) iff (2) means proving the following implication:

$$\begin{array}{ll} \text{if} & \mathcal{J} \models \Theta \\ \text{then} & \mathcal{J} \models \phi \text{ iff } \mathcal{J} \not\models \neg\phi. \end{array}$$

But the consequent is always true by definition of model of a negated formula. Therefore the implication is true. $\square$

In case $\Theta = \{\phi_1, \dots, \phi_n\}$, thanks to the above Lemma 4, we can decide whether $\Theta \models \phi$ or not by deciding on the unsatisfiability of $\Theta \cup \{(\neg\phi)\}$. This can be done by checking the unsatisfiability of $\bigwedge_{i=1}^n \phi_i \wedge (\neg\phi)$ with the truth table method (see Theorem 8).

Theorem 9 (Decidability 3). Let $\{\phi_1, \dots, \phi_n\} \cup \{\phi\}$ be a set of $\text{PL}(P)$ formulas.

- Deciding on $\{\phi_1, \dots, \phi_n\} \models \phi$ is equivalent to deciding on the unsatisfiability of $\bigwedge_{i=1}^n \phi_i \wedge (\neg\phi)$.
- There is an algorithm for deciding on $\{\phi_1, \dots, \phi_n\} \models \phi$. □

Example 12. Consider $\phi = p \vee q$ and $\Theta = \{p \vee q, p \vee \neg q\}$. We can decide on $\Theta \models \phi$ by means of truth tables in the following way. We can build the truth table of $(p \vee q) \wedge ((p \vee \neg q) \wedge \neg(p \vee q))$ for checking whether this formula is unsatisfiable: if yes, then $\Theta \models \phi$; if no, then $\Theta \not\models \phi$. Students are invited to do it.

Alternatively, we may argue over interpretations, by first observing that $\Theta = \{\phi, \phi'\}$. Now, $\mathcal{I} \models \Theta$ iff $\mathcal{I} \models \phi$ and $\mathcal{I} \models \phi'$. Therefore, if $\mathcal{I} \models \Theta$, then $\mathcal{I} \models \phi$.

Logical Equivalence

In case $\Theta = \{\psi\}$, that is, it contains only one formula, then we drop brackets and write $\psi \models \phi$. In particular, two formulae ψ and ϕ are **logically equivalent**, written as $\psi \equiv \phi$, if $\psi \models \phi$ and $\phi \models \psi$. In other words, two formula are semantically equivalent if, independently of their syntactic writing, they carry the same semantic information.

Theorem 10. $\phi \equiv \psi$ iff $(\phi \leftrightarrow \psi)$ is valid.

Proof. The equivalence follows from the definitions of $\equiv$ and of the interpretation of $\leftrightarrow$. □

Example 13. The following are all equivalent formulae:

$$\begin{array}{lll}
 p \rightarrow \perp & \equiv & \neg p; \\
 \neg\neg p & \equiv & p; & \text{Double Negation} \\
 p \vee q & \equiv & q \vee p; & \text{Commutativity of } \vee \\
 p \wedge q & \equiv & q \wedge p; & \text{Commutativity of } \wedge \\
 p \rightarrow q & \equiv & \neg p \vee q; \\
 p \leftrightarrow q & \equiv & (p \wedge q) \vee (\neg p \wedge \neg q); \\
 p \leftrightarrow q & \equiv & (p \rightarrow q) \wedge (q \rightarrow p).
 \end{array}$$

Substitutions

Deciding on whether two complex formulas are equivalent can be simplified by so-called substitutions. See also Section 3.7 of [Chiswell and Hodges, 2007] for a general introduction to substitutions, also relying on parsing trees.

Definition 4. Let ϕ, χ be formulae and p a propositional symbol of $\text{PL}(P)$. We define the **substitution** of ϕ by p in χ , written as $\chi[\phi/p]$, as follows:

- if $\chi = p$ then $\chi[\phi/p] = \phi$; if χ is any other atom then $\chi[\phi/p] = \chi$;
- if $\chi = (\neg\psi)$ then $\chi[\phi/p] = (\neg\psi[\phi/p])$;
- if $\chi = (\psi \square \psi')$ then $\chi[\phi/p] = (\psi[\phi/p] \square \psi'[\phi/p])$ for $\square \in \{\wedge, \vee, , \leftrightarrow\}$.

Example 14. Consider $\chi = p \wedge \neg q$. Then $\chi[p/q] = p \wedge \neg p$.

In the above example, notice that χ is satisfiable, whereas $\chi[p/q]$ is not. However, substitution preserves equivalence in the following sense: if two formulas χ and χ' are equivalent, the formulas $\chi[\phi/p]$ and $\chi'[\phi/p]$ are still equivalent. We give the following theorem without proof, which can be found in [Chiswell and Hodges, 2007], Section 3.7. An informal justification to the following statement is that, if we have a truth table validating $\chi \leftrightarrow \chi'$, by replacing p everywhere with ϕ , we obtain a truth table still validating $\chi[\phi/p] \equiv \chi'[\phi/p]$.

Theorem 11. If $\chi \equiv \chi'$ then $\chi[\phi/p] \equiv \chi'[\phi/p]$. □

Let us see an application.

Example 15. Consider $p \vee q$ and $q \vee p$. They are equivalent, as you can check by building the truth table of $p \vee q \leftrightarrow q \vee p$ and then appealing to Theorem 23, or reasoning about their respective models. By the above theorem, for any ϕ in $\text{PL}(P)$, we have $\phi \vee q \equiv q \vee \phi$. By the same theorem, for any ϕ' in $\text{PL}(P)$, we have $\phi \vee \phi' \equiv \phi' \vee \phi$.

Hereby is a list of equivalences, which can be proved as in the above example.

Commutativity	$\phi \vee \psi \equiv \psi \vee \phi$
	$\phi \wedge \psi \equiv \psi \wedge \phi$
	$\phi \leftrightarrow \psi \equiv \psi \leftrightarrow \phi$
Associativity	$(\phi \vee \psi) \vee \chi \equiv \phi \vee (\psi \vee \chi)$
	$(\phi \wedge \psi) \wedge \chi \equiv \phi \wedge (\psi \wedge \chi)$
Idempotence	$\phi \vee \phi \equiv \phi$
	$\phi \wedge \phi \equiv \phi$
Absorption	$\phi \vee (\phi \wedge \psi) \equiv \phi$
	$\phi \wedge (\phi \vee \psi) \equiv \phi$
	$\phi \vee \top \equiv \top$
	$\phi \wedge \perp \equiv \perp$
	$\phi \wedge \top \equiv \phi$
	$\phi \vee \perp \equiv \phi$
Distributivity	$\phi \wedge (\psi \vee \chi) \equiv (\phi \wedge \psi) \vee (\phi \wedge \chi)$
	$\phi \vee (\psi \wedge \chi) \equiv (\phi \vee \psi) \wedge (\phi \vee \chi)$
Double Negation	$\neg\neg\phi \equiv \phi$
De Morgan	$\neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi$
	$\neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi$
Others	$\phi \rightarrow \psi \equiv \neg\phi \vee \psi$
	$\phi \leftrightarrow \psi \equiv (\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)$
	$\phi \vee \neg\phi \equiv \top$
	$\phi \wedge \neg\phi \equiv \perp$

1.3 Proof Theory

1.3.1 Proof Calculi

Truth tables are easy to manage for formulae with few connectives and propositional symbols. However, the number of rows of the tables increases exponentially in the number of propositional symbols—cfr. Theorem 5. Secondly, truth tables cannot be extended beyond propositional connectives. Thirdly, truth tables do not unveil the *inference rules* for concluding that $\Theta \models \phi$. Such rules have the form

$$\frac{\text{formulae}}{\text{formulae}}$$

where the formula above the line are the **premises** of the rule, and those below the line are the **conclusions** of the rules. Informally, we say that a rule is *applied* to its premises and returns its conclusions.

Different rules give different proof calculi for $\text{PL}(P)$. Hereby, we overview two main such calculi:

- human oriented calculi, e.g., a natural deduction calculus with all the basic inference rules for concluding that $\Theta \models \phi$;
- computer oriented calculi, a.k.a. computational calculi, with fewer inference rules for deciding, as efficiently as possible, on the satisfiability of $\Theta \cup \{\neg\phi\}$, and hence dually on $\Theta \models \phi$ (see Lemma 4).

Natural Deduction Calculi (Optional)

A proof calculus of **natural deduction** $\mathcal{ND}$ for $\text{PL}(P)$ is meant to give all the basic inference rules for concluding that $\Theta \models \phi$. The inference rules of $\mathcal{ND}$ for $\text{PL}(P)$ are listed in Appendix A of [Chiswell and Hodges, 2007]. We already listed them, informally, and saw that they are everyday applied for reasoning about mathematical statements in natural language. $\mathcal{ND}$ turns them into formal inference rules for $\text{PL}(P)$.

The rules of $\mathcal{ND}$ allow us to build a derivation tree for a formula ϕ from formulae in Θ —notice that Θ can be empty. Curious students can read Definition 3.4.1 of derivation tree in p. 54 of [Chiswell and Hodges, 2007].

The fundamental result for $\mathcal{ND}$ is that it allows for verifying entailment and validity, consistently and completely, in the following sense.

Theorem 12. Let ϕ be a $\text{PL}(P)$ formula, and Θ a possibly empty set of $\text{PL}(P)$ formulae.

Soundness: if there is a derivation of ϕ from Θ in $\mathcal{ND}$ then $\Theta \models \phi$;

Completeness: if $\Theta \models \phi$ then there is a derivation of ϕ from Θ in $\mathcal{ND}$. □

Soundness is the easy part to prove by induction over the height of the derivation tree. Completeness requires some round-about, namely, constructing models for and out of sets of formulae with closure properties. Curious students can consult Section 3.10 of [Chiswell and Hodges, 2007].

Computational Calculi

There are two main types of computational calculi for $\text{PL}(P)$:

- calculi that are applicable to all propositional formulae, and can be easily extended to logics beyond propositional logic, e.g., tableau calculi;

- calculi that are based on and refine the truth table method, and are meant for specific propositional formulae, e.g., DPLL calculi.

Weakening the number or types of inference rules, however, has some costs: such calculi, in general, are not complete for entailment, as $\mathcal{ND}$ is.

However, in general, they are complete for (un)satisfiability in the following sense (a.k.a., **refutation complete**): if ϕ is unsatisfiable then the calculus allows for a derivation from $\neg\phi$ of $\perp$. More generally, if $\Theta \cup \{\neg\phi\}$ is unsatisfiable then the calculus allows for a derivation of $\perp$ from $\Theta \cup \{\neg\phi\}$ —see Lemma 4.

The following Subsection 1.3.2, which is mandatory for the course, focusses on a specific tableau calculus for $\text{PL}(P)$. This is the calculus we will employ in the course.

Then, Subsection 1.3.3 briefly overviews a DPLL calculus for $\text{PL}(P)$. This lays at the basis of the most used satisfiability procedure for $\text{PL}(P)$. DPLL is optional.

1.3.2 An Analytic Tableau Calculus

Analytic tableau calculi allow for rather efficient algorithms for deciding the satisfiability of a formula ϕ by systematically trying to build a model for ϕ . Dually, they aim at showing the validity of a formula ϕ by

- assuming that $\not\models \phi$, i.e., that there exists a model for $\neg\phi$,
- and proving that this assumption leads to a contradiction.

In the remainder, we first present the rules of such a calculus. We then present a procedure using the tableau rules for building a tableau for a formula, that is, a left-right labelled tree with the formula left labelling the root. We prove that a tableau so built can decide on the satisfiability of ϕ , and dually, it can decide on $\models \phi$. Finally, we show how tableaux can also be used to decide on the satisfiability of a set of formulae, and hence on entailment of a formula from a set of formulae. We confine ourselves to the case of finite sets, albeit tableaux can also be defined for infinite sets, e.g., see [Smullyan, 1968].

Tableau Rules

The proof rules of our analytic tableau calculus¹ are schematically presented in Table 1.1. Note that the premises of the tableau rules are complex formulae,

¹They are taken from [Smullyan, 1968], with ideas from [Bradley and Manna, 2007].

and never a propositional symbol or its negation. A propositional symbol p of P , or its negation, $\neg p$, is a **literal** of $\text{PL}(P)$; no other formula is a literal of $\text{PL}(P)$. Literals are usually denoted by l , with or without indices. Two literals are **complementary** if one is of the form p and the other of the form $\neg p$; in such a case, we write l and l^c . In some sense, to be specified below, literals do not contribute to the building of a tableau.

The tableau rules formalise the informal semantic arguments we applied so far when reasoning about whether an interpretation $\mathcal{I}$ satisfies a complex formula ϕ (that is, $\mathcal{I} \models \phi$), or $\mathcal{I}$ does not satisfy ϕ (that is, $\mathcal{I} \models \neg\phi$), according to the main connective of ϕ . Such arguments are schematically presented in Table 1.1.

Furthermore, tableau rules are partitioned into two classes of rules: α rules (or deterministic rules) that, when applied, extend a branch of the tableau; β rules (or nondeterministic rules) that, when applied, create a branching node and extend a branch of the tableau into two branches. See also the rule-type column in Table 1.1. Notice that α rules different than the $\neg\neg$ -rule have all the form

$$\frac{\alpha}{\frac{\alpha_1}{\alpha_2}}$$

whereas β rules have all the form

$$\frac{\beta}{\beta_1 \quad \beta_2}$$

Partitioning rules in two such big sets, that of α rules and that of β rules, will be pretty useful when proving theorems concerning tableau rules.

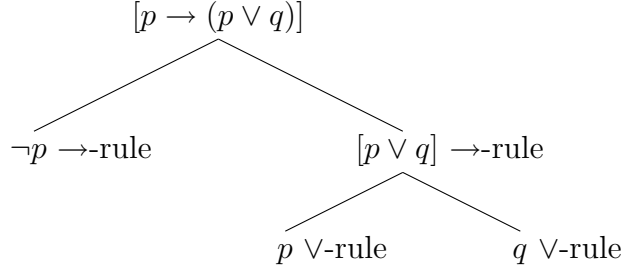
Tableau Procedure

Prologue. A **tableau** for ϕ is a left-right labelled binary tree $T = (N, D)$ with a left label and a right label functions. Its right labels are rule names, e.g., $\wedge$ -rule. Its left labels are $\text{PL}(P)$ formulae, possibly in square brackets. We first give some examples of tableaux for specific formulae, then define what a tableau for a generic formula is, as the result of a procedure.

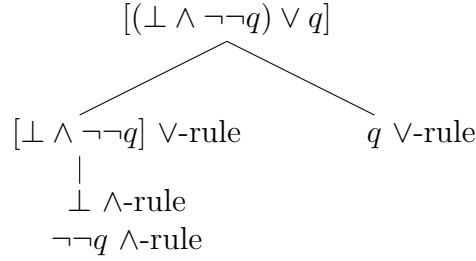
Example 16. The following is a tableau for $p \wedge q$ with α -rules only.

$$\begin{array}{c} [p \wedge q] \\ | \\ p \wedge\text{-rule} \\ q \wedge\text{-rule} \end{array}$$

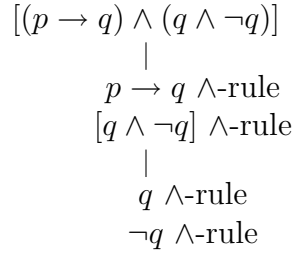
The following is a tableau for $p \rightarrow (p \vee q)$ with β -rules only.



The following is a tableau for $(\perp \wedge \neg\neg q) \vee q$ with α and β -rules.



The following is a tableau for $(p \rightarrow q) \wedge (q \wedge \neg q)$ with α -rules only.



In the following, a node is **unusable** if its left label is either a literal or in square brackets². Otherwise the node is **usable**. In the last tableau in the above Example 16, the node left labelled by $p \rightarrow q$ is usable; all the others are unusable, because they are left labelled by formulae that either are in square brackets or are literals. The idea is that, if a node is usable, a rule may still be applied to its left label, otherwise this is forbidden.

We distinguish the following types of branches of T : a branch of T is **closed** if two complementary literals left label its nodes, or $\perp$ left labels one of its nodes, or $\neg\top$ does it; otherwise the branch is **open**. The last tableau in the above Example 16 has only a closed branch. The first two tableau have only open branches. The third tableau has a closed branch (the left more one) and an open branch.

The following tableau procedure works on *usable* nodes of *open* branches of T .

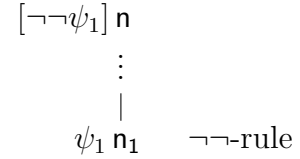
²Alternatively, cross out left labels that we enclose in brackets.

Procedure. The procedure starts with ϕ as left label of the root of T , without right label.

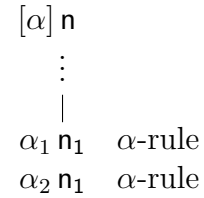
While there are usable nodes in open branches of T , choose one of them, say n . Enclose n in square brackets.

For every open branch of T passing through n , proceed by cases as follows.

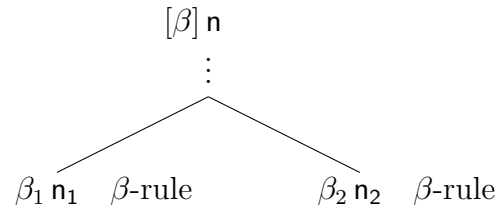
Case $\neg\neg$ -rule. If the left label of n is of the form $\neg\neg\psi_1$, then do as follows. Add a child n_1 to the leaf of branch, right label n_1 with $\neg\neg$ -rule, and left label n_1 with ψ_1 . See the figure at the right.



Case α -rule different than $\neg\neg$ -rule. If the left label of n is an α formula that does not have the form $\neg\neg\psi_1$, then do as follows. Add a child n_1 to the leaf of branch, right label n_1 with the appropriate α -rule, and left label n_1 with α_1 . Add a child n_2 to n_1 , right label n_2 with the appropriate α -rule, and left label n_2 with α_2 . See the figure at the right.



Case β -rule. If the left label of n is a β formula then do as follows. Add a left child n_1 and a right child n_2 to the leaf of branch; right label n_1 and n_2 with the appropriate β -rule. Left label n_1 with β_1 , and left label n_2 with β_2 . See the figure at the right.



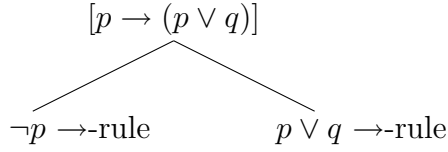
Note 5. Note that we do not write the edge from n_1 to n_2 , created by an application of an α rule different than the $\neg\neg$ -rule. Such a notational expedient allows us to see, at a glance, that both nodes are obtained by the same rule application. Moreover, in the examples, we do not write down the names of the nodes of the tableaux, as they are rather unnecessary.

The tableaux in Example 16 were built according to the above procedure. Let us see how the procedure builds the first three tableaux, at each iteration of the while loop.

Example 17. The tree built before entering the while-loop is just the root labelled by $p \wedge q$. At the first iteration of the loop, the root is chosen and the formula left

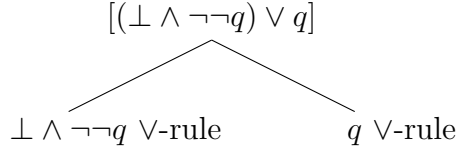
labelling it is enclosed in square brackets. Since this formula is an α formula, the α -rule for $\wedge$ is applied. The first tableau of the previous example is produced, and then the procedure terminates (all nodes are unusable).

Let us consider the second formula of the above example. The tree built before entering the while-loop is just the root labelled by $p \rightarrow (p \vee q)$. At the first iteration of the loop, the root is chosen and the formula left labelling it is enclosed in square brackets. Since this formula is a β formula, the β -rule for $\rightarrow$ is applied, producing the following tableau.



At the second iteration of the while loop, the node labelled by $p \vee q$ is chosen—the only usable node of T , since $\neg p$ is a literal and the formula labelling the root is in square brackets. This node is enclosed in square brackets, and the $\vee$ -rule is applied. The second tableau of the previous example is produced, and then the procedure terminates (all nodes are unusable).

Let us consider the third formula of the previous example. The tree built before entering the while-loop is just the root labelled by $(\perp \wedge \neg\neg q) \vee q$. At the first iteration of the loop, the root is chosen and the formula left labelling it is enclosed in square brackets. Since this formula is a β formula, the β -rule for $\rightarrow$ is applied, producing the following tableau.



At the second iteration of the while loop, the node labelled by $\perp \wedge \neg\neg q$ is chosen—the only usable node of T , since q is a literal and the formula labelling the root is in square brackets. This node is enclosed in square brackets, and the $\wedge$ -rule is applied. The third tableau of the previous example is produced, and then the procedure terminates (all nodes in *open* branches are unusable).

Termination. The only commitment we have on the node n chosen (guessed) in the while loop is that n is usable in an open branch. In this manner, the above procedure always terminates because of the following two reasons.

1. Once a node n is chosen, it is placed in square brackets and hence never chosen again in the while loop. This means that the while loop eventually terminates if the for loop terminates.

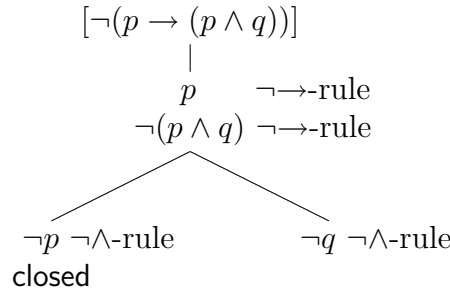
2. For each left label of a node, let: $L(\text{node})$ be equal to the number of occurrences of logical symbols different than $\leftrightarrow$ in the left label of **node**; $B(\text{node})$ be equal to 1 plus the number of occurrences of $\leftrightarrow$ in the left label of **node**. Take **branch** considered for extension in an iteration of the for-loop, and terminating with **leaf**; let **node** be any of the (finitely many) nodes added to **branch** at the end of the iteration. Then $B(\text{node}) + L(\text{node}) < B(\text{leaf}) + L(\text{leaf})$. In other words, at each iteration, the natural number $B(\text{node}) + L(\text{node})$ decreases. Therefore the for loop eventually terminates.

Upon termination, each branch is either closed, or open with only unusable nodes. We will call such branches **finished**. We have just proved the following theorem.

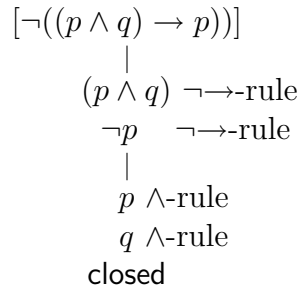
Theorem 13 (Termination). The above algorithm with input ϕ terminates returning a tableau for ϕ , each branch of which is finished. $\square$

Note 6. While building a tableau, it is convenient to mark its closed branches as in the following examples, so as to distinguish them at a glance from open branches—as the while and for loops of the tableau procedure do *not* consider closed branches.

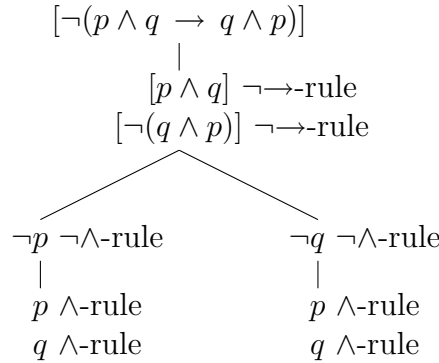
Example 18. The following is a tableau for $\neg(p \rightarrow (p \wedge q))$. It has an open branch, and a closed branch.



The following is a tableau for $\neg((p \wedge q) \rightarrow p)$. All its branches are closed.



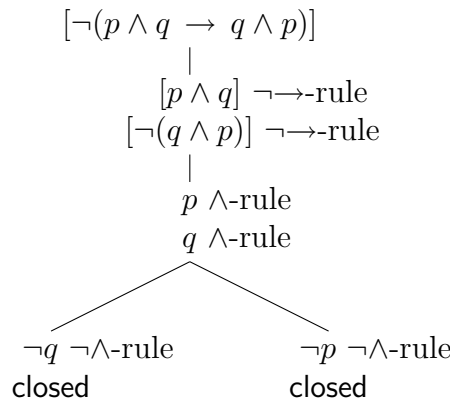
The following is a tableau for $\neg((p \wedge q) \rightarrow (q \wedge p))$. All its branches are closed.



Note also that, in the previous tableau, the applied $\neg \rightarrow$ -rule returns two formulae, one of which is an α -formula and the other is a β -formula. The tableau continues by first choosing the node labelled by the β -formula $\neg(q \wedge p)$, but we could make a different choice: we could first choose the node labelled by the α -formula $p \wedge q$ —try building this tableau before continuing reading. $\square$

Note 7. The choice of n in the while loop can be driven by optimisation heuristics. For instance, one can always choose first nodes with α formulae, and then nodes with β formulae. In this manner, the same formulae are not repeated in different branches; rather, they occur as left labels above all those branching nodes. Let us see an example of a tableau built in this manner.

Example 19. The following is a tableau for $\neg(p \wedge q \rightarrow q \wedge p)$ following the optimisation heuristics. Compare it with the tableau for the same formula given in the previous example.



Satisfiability and Validity via Tableaux

In the following, we prove that every satisfiable formula has a tableau with an open branch. We will say that $\mathcal{I}$ **satisfies a branch** of a tableau if $\mathcal{I}$ satisfies *all* the formulae left labelling nodes of the branch.

Theorem 14.

1. If $\mathcal{J} \models \phi$, then there is an open branch satisfied by $\mathcal{J}$ in any tableau for ϕ .
2. If a tableau for $\neg\phi$ has only closed branches, then $\models \phi$ (**soundness**).

Proof.

(1) Consider a tableau for ϕ . We prove the following (invariant) statement by induction over the iterations of the for-loop of the tableau procedure:

there exists an open branch of the tableau that $\mathcal{J}$ satisfies.

Clearly, ϕ , left labelling the root, is satisfied by its model $\mathcal{J}$. Let us now consider the n -th iteration of the for loop, with $n \neq 0$. Take the open branch satisfied by $\mathcal{J}$ and extended in the iteration. Each formula that left labels a new leaf is introduced by an α -rule or a β -rule. Therefore, by the informal semantic argument in Table 1.1, at least one of the extended branches is satisfied by $\mathcal{J}$. In this manner, we proved that there exists a branch in any tableau for ϕ that $\mathcal{J}$ satisfies. Clearly, such a branch is also open—since $\mathcal{J}$ cannot satisfy $\perp$, $\neg\top$ or complementary literals. This concludes the proof of the first statement.

(2) The first statement yields the following:

if $\neg\phi$ is satisfiable,
then there exists an open branch in any tableau for $\neg\phi$.

This means that, if there exists *no* open branch in a tableau for $\neg\phi$, then $\neg\phi$ is unsatisfiable. In other words, for all $\mathcal{J}$, $\mathcal{J} \not\models \neg\phi$. Equivalently, for all $\mathcal{J}$, $\mathcal{J} \models \phi$, which means that ϕ is valid. $\square$

Next, we prove that an open branch of a tableau for ϕ defines a model for ϕ . Completeness of the calculus follows from this.

Theorem 15.

- Any open branch of a tableau for ϕ defines an interpretation $\mathcal{J}$ that satisfies the branch, and in particular $\mathcal{J}$ satisfies ϕ .
- If $\models \phi$ then any tableau for $\neg\phi$ has only closed branches (**completeness**).

Proof.

(1) Consider an open branch of a tableau. For each $p \in P$ do as follow: if p occurs as left label in the branch, set $\mathcal{I}(p) = \mathbf{T}$; if $\neg p$ occurs as left label, set $\mathcal{I}(p) = \mathbf{F}$; else set any arbitrary value for $\mathcal{I}(p)$. The proof that $\mathcal{I}$ satisfies the branch (that is, every formula occurring as left label in the branch) goes by induction on ϕ occurring as left label of the branch.

If ψ is an atomic formula of the branch, then $\mathcal{I} \models \psi$ by definition of $\mathcal{I}$.

Let ψ be a complex formula of the branch. Consider the case $\psi = (\psi_1 \wedge \psi_2)$. Case $\wedge$ of the tableau construction ensures that both ψ_1 and ψ_2 are in the branch. By induction hypothesis, both $\mathcal{I} \models \psi_1$ and $\mathcal{I} \models \psi_2$ hold, therefore $\mathcal{I} \models \psi$ holds as well. Next, consider the case $\psi = \neg(\psi_1 \wedge \psi_2)$. Case $\neg\wedge$ of the tableau construction ensures that either $\neg\psi_1$ or $\neg\psi_2$ is in the branch; let us say that $\neg\psi_1$ is in the branch. By induction hypothesis, $\mathcal{I} \models \neg\psi_1$ holds, therefore $\mathcal{I} \models \psi$ holds as well. The other cases are treated similarly.

This allows us to conclude that $\mathcal{I}$ satisfies the branch, that is, all the formulae left labelling its nodes. In particular, $\mathcal{I}$ satisfies the formula ϕ that left labels the root of the branch.

(2) Let us consider ϕ so that $\models \phi$. Then $\neg\phi$ is unsatisfiable, that is, there is *no* interpretation that satisfies $\neg\phi$. Let us build a(ny) tableau for $\neg\phi$. By the first statement, if the tableau has an open branch, then there exists an interpretation that satisfies $\neg\phi$. But this is impossible. Therefore all tableaux for $\neg\phi$ have only closed branches. $\square$

A tableau for $\neg\phi$ with only closed branches is a **derivation** from ϕ of $\perp$. In this sense, Theorems 15 and 14 state that the tableau calculus is refutation complete and sound, respectively. Thanks to those theorems and the termination Theorem 13, we can use tableaux for deciding on satisfiability and validity for $\text{PL}(P)$.

Corollary 1 (Decidability 4). Consider any $\phi \in \text{PL}(P)$. The following are decision procedures.

(satisfiability) Build a tableau for ϕ with the procedure in page 24: if the tableau for ϕ has an open branch, then ϕ is satisfiable, and the open branch defines a model for ϕ as in the proof of Theorem 15; else ϕ is unsatisfiable.

(validity) Build a tableau for $\neg\phi$ with the procedure in page 24: if all the branches of the tableau for $\neg\phi$ are closed, then ϕ is valid; else ϕ is falsifiable, and an open branch of the tableau defines an interpretation that does not satisfy ϕ as in the proof of Theorem 15. $\square$

Let us see examples for both.

Example 20 (Verifying Validity with Tableaux). Let us prove that $\models (q \rightarrow (p \rightarrow q))$ by building a tableau for $\neg(q \rightarrow (p \rightarrow q))$ with only closed branches. We will not right label nodes, nor enclose them in square brackets. Students are asked to do it as exercise.

$$\begin{array}{c}
 \neg(q \rightarrow (p \rightarrow q)) \\
 | \\
 q \\
 \neg(p \rightarrow q) \\
 | \\
 p \\
 \neg q \\
 \text{closed}
 \end{array}$$

Let us prove that $\not\models (q \vee p)$ by building a tableau for $\neg(q \vee p)$ with an open branch.

$$\begin{array}{c}
 \neg(q \vee p) \\
 | \\
 \neg q \\
 \neg p
 \end{array}$$

The formulae $\neg p$ and $\neg q$ in the (only) open branch give the interpretation $\mathcal{I}(p) = \mathcal{I}(q) = \mathbf{F}$ that falsifies $q \vee p$ (i.e., is not a model of $p \vee q$).

Example 21 (Verifying Satisfiability with Tableaux). Let us prove that $(p_{11} \wedge \neg p_{12}) \vee (\neg p_{11} \wedge p_{12})$ is satisfiable by building a tableau for $(p_{11} \wedge \neg p_{12}) \vee (\neg p_{11} \wedge p_{12})$ with an open branch (at least).

$$\begin{array}{c}
 (p_{11} \wedge \neg p_{12}) \vee (\neg p_{11} \wedge p_{12}) \\
 \swarrow \quad \searrow \\
 (p_{11} \wedge \neg p_{12}) \quad (\neg p_{11} \wedge p_{12}) \\
 \begin{array}{cc}
 | & | \\
 p_{11} & \neg p_{11} \\
 \neg p_{12} & p_{12}
 \end{array}
 \end{array}$$

This tableau has two open branches, and hence it gives two distinct models for $(p_{11} \wedge \neg p_{12}) \vee (\neg p_{11} \wedge p_{12})$:

- $\mathcal{J}_1$ with $\mathcal{J}_1(p_{11}) = \mathbf{T}$ and $\mathcal{J}_1(p_{12}) = \mathbf{F}$;
- $\mathcal{J}_2$ with $\mathcal{J}_2(p_{11}) = \mathbf{F}$ and $\mathcal{J}_2(p_{12}) = \mathbf{T}$.

Let us prove that $(q \wedge \neg q)$ is unsatisfiable by building a tableau for $(q \wedge \neg q)$ with only closed branches.

$$\begin{array}{c}
q \wedge \neg q \\
| \\
q \\
\neg q \\
\text{closed}
\end{array}$$

Satisfiability of a Set of Formulae and Entailment via Tableaux

Hereby we show how we can build a tableau for deciding on: (1) the satisfiability of a finite set Θ of formulae; (2) $\Theta \models \phi$ with Θ finite.

First of all, let us recall the following reductions from Section 1.2:

Theorem 8: deciding on the satisfiability of $\{\phi_1, \dots, \phi_n\}$ is equivalent to deciding on the satisfiability of $\bigwedge_{i=1}^n \phi_i$;

Theorem 9: deciding on $\{\phi_1, \dots, \phi_n\} \models \phi$ is equivalent to deciding on the unsatisfiability of $\bigwedge_{i=1}^n \phi_i \wedge (\neg\phi)$.

Such theorems and Corollary 1 allow us to use tableaux for deciding on the satisfiability of Θ or $\Theta \models \phi$, with $\Theta = \{\phi_1, \dots, \phi_n\}$.

Corollary 2 (Decidability 5). Consider any $\{\phi_1, \dots, \phi_n\} \cup \{\phi\} \subseteq \text{PL}(P)$. The following are decision procedures.

(satisfiability) Build a tableau for $\bigwedge_{i=1}^n \phi_i$ with the procedure in page 24. If the tableau has an open branch, then $\{\phi_1, \dots, \phi_n\}$ is satisfiable, and the open branch defines a model for the set; else the set is unsatisfiable.

(entailment) Build a tableau for $\bigwedge_{i=1}^n \phi_i \wedge (\neg\phi)$ with the procedure in page 24. If all the branches of the tableau are closed, then $\{\phi_1, \dots, \phi_n\} \models \phi$; else $\{\phi_1, \dots, \phi_n\} \not\models \phi$, and an open branch of the tableau defines a model of $\{\phi_1, \dots, \phi_n\}$ that does not satisfy ϕ . $\square$

Example 22 (Verifying Set Satisfiability with Tableaux). Let us prove that the set $\{p \vee q, \neg p\}$ is satisfiable by building a tableau with an open branch for the formula $(p \vee q) \wedge \neg p$.

$$\begin{array}{c}
(p \vee q) \wedge \neg p \\
| \\
p \vee q \\
\neg p \\
\swarrow \quad \searrow \\
p \quad q \\
\text{closed}
\end{array}$$

The right-more branch gives the interpretation $\mathcal{I}(q) = \mathbf{T}$ and $\mathcal{I}(p) = \mathbf{F}$ that satisfies the set.

Note 8. In general, to save space, we can silently apply the $\wedge$ -rule when building a tableau for $\Theta = \{\phi_1, \dots, \phi_n\}$, and write all the formulae of Θ one below the other, left labelling the root. With such expedient, the previous tableau would be shortened as follows.

$$\begin{array}{c} p \vee q \\ \neg p \\ \swarrow \quad \searrow \\ p \quad q \\ \text{closed} \end{array}$$

Example 23 (Verifying Entailment with Tableaux). Let us prove $p \wedge q \models p$ by building a tableau for $(p \wedge q) \wedge \neg p$ with only closed branches.

$$\begin{array}{c} (p \wedge q) \wedge \neg p \\ | \\ p \wedge q \\ \neg p \\ | \\ p \\ \text{closed} \end{array}$$

Note 9. With the expedient in Note 8, the previous tableau can be shortened as follows.

$$\begin{array}{c} p \wedge q \\ \neg p \\ | \\ p \\ \text{closed} \end{array}$$

1.3.3 The DPLL Calculus (Optional for the Course)

Specific propositional logic calculi can be developed for formulae that combines literals only with $\wedge$ and $\vee$. If l denotes a literal of $\text{PL}(P)$, a **clause** of $\text{PL}(P)$ is defined as follows:

$$C, C' \longrightarrow l \mid \perp \mid \top \mid C \vee C'.$$

Example 24. The formulae p , $\top$, $\neg$ and $p \vee \top \vee \neg p$ are clauses of $\text{PL}(\{p\})$.

A formula θ of $\text{PL}(P)$ is in **Conjunctive Normal Form** (CNF) if it is a *conjunction* of clauses: $\bigwedge_{i=1}^n C_i$.

Rule	Rule name	Rule type	Semantic Argument
$\frac{\psi_1 \wedge \psi_2}{\psi_1 \quad \psi_2}$	$\wedge$ -rule	α	if $\mathcal{J} \models \psi_1 \wedge \psi_2$ then $\mathcal{J} \models \psi_1$ and $\mathcal{J} \models \psi_2$
$\frac{\neg(\psi_1 \wedge \psi_2)}{\neg\psi_1 \quad \neg\psi_2}$	$\neg\wedge$ -rule	β	if $\mathcal{J} \models \neg(\psi_1 \wedge \psi_2)$ then $\mathcal{J} \models \neg\psi_1$ or $\mathcal{J} \models \neg\psi_2$
$\frac{\psi_1 \vee \psi_2}{\psi_1 \quad \psi_2}$	$\vee$ -rule	β	if $\mathcal{J} \models \psi_1 \vee \psi_2$ then $\mathcal{J} \models \psi_1$ or $\mathcal{J} \models \psi_2$
$\frac{\neg(\psi_1 \vee \psi_2)}{\neg\psi_1 \quad \neg\psi_2}$	$\neg\vee$ -rule	α	if $\mathcal{J} \models \neg(\psi_1 \vee \psi_2)$ then $\mathcal{J} \models \neg\psi_1$ and $\mathcal{J} \models \neg\psi_2$
$\frac{\psi_1 \rightarrow \psi_2}{\neg\psi_1 \quad \psi_2}$	$\rightarrow$ -rule	β	if $\mathcal{J} \models \psi_1 \rightarrow \psi_2$ then $\mathcal{J} \models \neg\psi_1$ or $\mathcal{J} \models \psi_2$
$\frac{\neg(\psi_1 \rightarrow \psi_2)}{\psi_1 \quad \neg\psi_2}$	$\neg\rightarrow$ -rule	α	if $\mathcal{J} \models \neg(\psi_1 \rightarrow \psi_2)$ then $\mathcal{J} \models \psi_1$ and $\mathcal{J} \models \neg\psi_2$
$\frac{\psi_1 \leftrightarrow \psi_2}{\psi_1 \wedge \psi_2 \quad \neg\psi_1 \wedge \neg\psi_2}$	$\leftrightarrow$ -rule	β	if $\mathcal{J} \models \psi_1 \leftrightarrow \psi_2$ then $\mathcal{J} \models \psi_1 \wedge \psi_2$ or $\mathcal{J} \models \neg\psi_1 \wedge \neg\psi_2$
$\frac{\neg(\psi_1 \leftrightarrow \psi_2)}{\psi_1 \wedge \neg\psi_2 \quad \neg\psi_1 \wedge \psi_2}$	$\neg\leftrightarrow$ -rule	β	if $\mathcal{J} \models \neg(\psi_1 \leftrightarrow \psi_2)$ then $\mathcal{J} \models \psi_1 \wedge \neg\psi_2$ or $\mathcal{J} \models \neg\psi_1 \wedge \psi_2$
$\frac{\neg\neg\psi_1}{\psi_1}$	$\neg\neg$ -rule	α	if $\mathcal{J} \models \neg\neg\psi_1$ then $\mathcal{J} \models \psi_1$

Table 1.1: Tableau Rules.

$\frac{l \quad C(l^c)}{C[\perp/l^c]} \text{ unit resolution}$	$\frac{\theta(l)}{\theta[\top/l] \quad \theta[\perp/l]} \text{ splitting}$
---	--

Figure 1.3: The unit propagation and splitting rules for DPLL.

Example 25. The formulae, p , $(p \wedge (\neg q))$, $((p \wedge (\neg q)) \vee (q \wedge (\neg r) \wedge (\neg s \wedge \neg s)))$ are in CNF.

Notice that verifying the validity of a CNF formula is an easy task, that can be done in time polynomial in the number n of literals and in the number m of clauses of the formula (in $\Theta(n^2m)$, worst case).

Theorem 16. A CNF formula θ is valid iff every clause of θ has a pair of complementary literals.

Proof. Clearly, if every clause of θ has a pair of complementary literals, then every clause is true in any interpretation. Therefore θ is true in any interpretation.

Let us now prove the vice versa reasoning by contraposition. Assume that C is a clause of θ with no complementary literals. For simplicity of exposition, assume that $C = l_1 \vee l_2$. Since l_1 and l_2 are not complementary, let $\mathcal{I}$ be the interpretation that is a model of neither l_1 nor l_2 . Therefore $\mathcal{I}$ is not a model of θ . Therefore θ is not valid. $\square$

Now, is there an algorithm for transforming *every* formula of $\text{PL}(P)$ into an *equivalent* formula in CNF or DNF? The answer is positive, albeit the transformation, based on the list of equivalences in Section 1.2, may produce a CNF formula whose size is exponential in the size of the original one.

A more efficient transformation can be achieved in the following manner. Instead of returning an equivalent formula, the transformation returns an **equi-satisfiable** formula: ϕ is satisfiable iff its transformed formula is satisfiable. For more on this, see [Huth and Ryan, 2010] or [Bradley and Manna, 2007]. However, verifying the satisfiability of a CNF formula is difficult.

Theorem 17 (CookLevin). Deciding on the satisfiability of a formula in conjunctive normal form, where each clause is even limited to at most three literals, is NP-complete. $\square$

For a proof of the the Cook-Levin theorem, see the *Theory of Computing* course.

The Davis Putnam Longeman Lovelace (DPLL) calculus consists (mainly) of the rules in Figure 1.3 for soundly verifying the (un)satisfiability of a CNF formula—in other words, the calculus is refutation complete and sound.

- Splitting, a nondeterministic rule, is based on the space-efficient idea of choosing (guessing) and building only one row of the truth table at a time. Given θ , it chooses a literal l of θ : in one case $\top$ substitutes l (interpreting this as true), in the other $\perp$ substitutes l (interpreting this as false).
- Unit resolution, a deterministic rule, exploits the CNF form. The premises of unit resolution are a clause consisting of precisely one literal l , and a clause C of which l^c is a disjunct. Then $\perp$ substitutes l^c in C .

DPLL algorithms decide on whether a CNF formula is satisfiable or not by scheduling those rules—and possibly other optimisation rules. For instance, the rules can be scheduled according to the following strategy:

- for all unit clauses, unit resolution is applied;
- if unit resolution is no more applicable, then splitting is applied.

A recursive version of the DPLL algorithm following this strategy is proposed, for instance, in [Bradley and Manna, 2007].

1.4 Summary

We saw that propositional logic has: a language with its own grammar; a semantics; a proof theory.

1.4.1 Language

$\text{PL}(P)$ is the propositional logic language over P . For every formula of $\text{PL}(P)$ we can generate a unique parsing tree; we can use such parsing trees to recognise whether an expression over P is a formula or not.

1.4.2 Semantics

- We defined interpretations $\mathcal{I}$ as functions from P to $\{\text{T}, \text{F}\}$. Each interpretation can be uniquely extended over $\text{PL}(P)$, which allows us to define $\mathcal{I} \models \phi$ for a generic $\phi \in \text{PL}(P)$; one way of reading this is “ $\mathcal{I}$ satisfies ϕ ”.

- We defined when a formula is satisfiable, unsatisfiable, valid (tautology), falsifiable. We defined a procedure for deciding whether a formula is satisfiable/unsatisfiable, valid/falsifiable by using truth tables.
- We extended the notion of satisfiability to sets of formulae. We reduced the satisfiability of a finite set of formulae to the satisfiability of their conjunction. Hence, any decision procedure for the latter can be used for the former.
- We extended the notion of validity to entailment, and reduced entailment of a formula from a finite set of formulae to satisfiability of a finite set of formulae. Hence, any decision procedure for the latter can be used for the former.

1.4.3 Proof Theory

- We defined the notion of proof calculus in general, and distinguished between: natural deduction calculi, complete for entailment; computational calculi, refutation complete.
- We overviewed a natural deduction calculus and explained its rationale (optional).
- We introduced the analytic tableau calculus.
 - We proved that it is a refutation sound and complete calculus for validity.
 - We proved that it gives us a procedure for deciding on: the satisfiability or not of ϕ by building a tableau for ϕ ; the validity or not of ϕ by building a tableau for $(\neg\phi)$.
 - We extended this procedure for deciding on the satisfiability of a finite set of formulae, and hence of the entailment of a formula from finitely many formulae.
- We overviewed a DPLL calculus (optional).

Chapter 2

First Order Logic

Propositional logic allows us to reason with connectives. If we can conclude that $\Sigma \models \phi$ in propositional logic (e.g., via the tableau procedure), then we are sure that the sentence formalised by ϕ “follows logically” from the sentences formalised by Σ . However, consider the following argument (syllogism).

All men are mortals.
Socrates is a men.
Therefore Socrates is mortal.

It seems a perfectly logical argument. However, propositional logic does not allow us to formalise it and conclude that “Socrates is a man” follows logically from its premises.

What any propositional language is missing are a universal quantifier, variables and (symbols for) properties that allow us to render “all things that are men”, independently of the number of men (three, three million or an indefinite number). First order languages enrich propositional languages with those—and even more.

2.1 Language

As in the case of a propositional language, we first define the lexicon of a first order language, then its expressions and finally its formulae. Study also Section 5.3 of your textbook, excluding Definitions 5.3.4 and 5.3.7.

2.1.1 Lexicon and Expressions

The first order symbols are all propositional symbols plus variables, countably many, and two quantifiers.

Definition 5. The first-order symbols are distinct

- countably many **variables**, denoted by x, y, z , possibly with indices,
- the propositional parentheses,
- the propositional connectives,
- $\forall$ (**universal quantifier**), and $\exists$ (**existential quantifier**).

The symbol $\forall$ is used to express universal claims, those we express in English using quantified phrases with “everything”, “each thing”, “all things”, “anything” etc. It is always used in connection with a variable, followed by a formula. The combination $\forall x$ is read “for every object x ” or “for all x ”. Then $\forall xP(x)$ will be read, for instance, as “for all x with the property P ”.

Dually, the symbol $\exists$ is used to express existential claims, those we express in English using quantified phrases with “something”, “at least a thing”, “there is a thing”, “there exists a thing” etc. It is always used in connection with a variable as well, followed by a formula. The combination $\exists x$ is read “there exists an object x ” or “there is x ”. Then $\exists xP(x)$ will be read, for instance, as “there is x with the property P ”.

Definition 6. A first-order **signature** σ has distinct:

- constant symbols, denoted by c, d possibly with indices;
- function symbols, denoted by f, g possibly with indices, each with its own arity $n > 0$;
- relation or predicate symbols, denoted by P, R possibly with indices, each with its own arity $n > 0$;
- possibly, the (binary) equality symbol $=$.

Individual constants are simply symbols that are used to refer to some fixed individual object. They are the analogue of proper names, like Dario, or numbers, like 1. However, the analogy with proper names stops there: an individual constant refers to one object of our domain of discourse, and no more.

Function symbols are used to refer to noun phrases, like “Dario’s father”, or functions, like $|x|$. The arity of a function symbols fixes the number of objects that the function applies to, e.g., one in case of the function symbol for the absolute value.

Relation symbols express some property of objects or some relation between objects, like “likes” between two people, $<$ between numbers, “blue” of objects. The arity of a relation symbols fixes the number of objects involved in the relation, e.g., two in case of $<$, one in case of “blue”.

Definition 7. The **lexicon** of σ precisely contains all first-order symbols and σ .

Study the lexicon of Example 5.3.2 of your textbook, p. 112.

As in the case of the propositional language, we use the lexicon for building expressions, like in the propositional case.

Definition 8. An **expression** or **word** of σ is a finite sequence of elements of the lexicon of σ .

The *occurrence* of a symbol in an expression is defined as in the propositional case.

2.1.2 Terms and Formulae

As in the case of propositional formulae, complex first order formulae are built out of atomic formulae by means of finitely many applications of connectives and also quantifiers; we will focus on a specific class of formulae, without ‘meaningless variables’, namely, the so-called closed formulae.

However, the atomic formulae of a first order language are no more propositional symbols but expressions built with relation symbols out of so-called terms.

Terms

Terms are variables, constant symbols, function symbols and their combination to formalise noun phrases, e.g., proper names and pronouns within our domain of discourse. Examples of terms are “Dario”, “he”, “2”, “ x ”, “the square root of 2”, “the square root of x ”.

Formally, terms are inductively defined as follows.

A **term** of σ , denote by s or t , possibly with indices, is:

- a variable,

- a constant symbol of σ ,
- the expression $f(t_1, \dots, t_n)$, with $t_1, \dots, t_n$ terms of σ and f an n -ary function symbol of σ , with $n > 0$;
- no other expression of σ is a term of σ .

Example 26. The following are examples of terms for σ_0 , with f a unary function symbol and c a constant symbol: $c, f(c), f(x), f(f(x))$. The following are examples of terms for σ_1 that has one more function symbol than σ_0 , namely, g , a binary function symbol: $c, g(f(c), c), g(f(x), c), g(f(f(c)), x)$.

As in the case of propositional formulae, each term is generated by (associated to) a *unique parsing tree for terms* of σ : the leaves of the tree are right labelled by variables or constant symbols of σ ; a node of the tree with n children, with $n > 0$, is right labelled by an n -ary function symbol of σ . The parsing left labelling of the tree is defined as in Figure 2.1, taken from [Chiswell and Hodges, 2007], p. 116: α denotes a constant symbol, F an n -ary function symbol.



Figure 2.1: Parsing left labelling for σ terms.

Study also the example of a parsing tree associated to a term of the signature of arithmetics in p. 114 of your textbook.

The proof of uniqueness is similar to that for propositional formulae.

Theorem 18 (Unique readability for terms.). An expression is a term of σ iff it is associated to a unique parsing tree for terms of σ . $\square$

This allows us to use parsing trees for σ -terms not only for generating terms but also for recognising whether an expression is, or not, a term.

Example 27. Consider σ_1 as above. Then $f(f(c))$ is a term whose parsing tree has three nodes: the root is left labelled by $f(f(c))$; its child is left labelled by $f(c)$; the leaf is left labelled by c . Consider σ_1 as above. Then $g(f(c), x)$ is associated to a binary parsing tree: its root is left labelled by $g(f(c), x)$; the two children, l and r , are left labelled by $f(c)$ and x , respectively; l has a child that is left labelled by c . On the other hand, $f(f(f))$ is not a term because f is neither a constant symbol nor a variable.

Formulae

Formulae allow us to formalise complex sentences about objects of our discourse domain. Examples are “if all men are mortal and Socrates is a man, then Socrates is mortal” or sentences defining total orders (or, axioms) like “for all x and y , either $x < y$ or $y < x$ ”.

Let σ be a first-order language without $=$. A **first order formula** of σ , denote by ϕ or ψ , possibly with indices, is:

- an **atomic formula** of the form $R(t_1, \dots, t_n)$, where $t_1, \dots, t_n$ are terms of σ , and R an n -ary relation symbol of σ ;
- $(\neg\phi)$ and $(\phi \square \psi)$, where $\square$ is any binary connective, ϕ and ψ are first-order formulae of σ ;
- $\forall x\phi$ where x is a variable and ϕ is a first order formula of σ ;
- $\exists x\phi$ where x is a variable and ϕ is a first order formula of σ ;
- no other expression of σ is a term of σ .

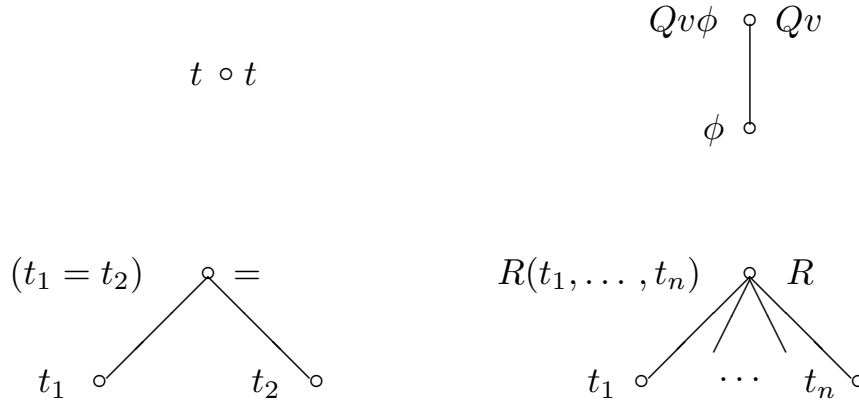
If σ has $=$ then an expression of the form $t = s$, where t and s are terms of σ , is also an atomic formula of σ .

The σ *first order language*, denoted by $\text{FOL}(\sigma)$, is the set of first order formula of σ .

Example 28. The following are examples of atomic formulae for σ_2 , with f a unary function symbol, c a constant symbol, and P a unary relation symbol: $P(c)$, $P(x)$, $P(f(c))$, $P(f(x))$, $P(f(f(x)))$. The following are examples of atomic formulae for σ_3 that has one more relation symbol than σ_2 , namely, R , a binary function symbol: c , $R(f(c), c)$, $R(f(x), c)$, $R(f(f(c)), x)$. Examples of formulae (non atomic, that is, **complex**) of σ_1 are: $(R(f(c), c) \wedge R(f(x), c))$, $\forall x R(f(x), c)$, $\forall x (R(f(c), c) \wedge R(f(x), c))$, $(R(f(c), c) \wedge \forall x R(f(x), c))$.

As in the case of propositional formulae, a first order formula of σ is generated by (associated to) a *unique* parsing tree for σ formulae. In your textbook, study Definition 5.3.7 of **parsing trees for first order formulae** of σ . Their parsing left labelling is like the one for propositional formulae extended with the cases 2.2, taken from [Chiswell and Hodges, 2007], p. 116: Q is a universal or existential quantifier, R is an n -ary relation, t is any term.

Study also the example of a parsing tree associated to a formula of the signature of arithmetics in p. 116 and 117 of your textbook.

Figure 2.2: Parsing left labelling for $\text{FOL}(\sigma)$.

The proof of the uniqueness readability theorem is similar to that for propositional formulae.

Theorem 19 (Unique readability for first order formulae.). An expression is a first formula of σ iff it is associated to a unique parsing tree for formulae of σ . $\square$

This allows us to use parsing trees for σ -formulae not only for generating formulae but also for recognising whether an expression is, or not, a first order formula.

Example 29. Consider σ_3 as above. Then $P(f(c))$ is an atomic formula whose parsing tree has two nodes: the root is left labelled by $P(f(c))$; its child is left labelled by $f(c)$, that is a term—build its parsing tree, separately. Then $R(c, x)$ is associated to a binary parsing tree: its root is left labelled by $R(c, x)$; its two children (and leaves), l and r , are left labelled by c and x , respectively, that are terms. On the other hand, $P(P(x))$ is not an atomic formula because $P(x)$ is not a term. The expression $\forall x P(x)$ is a formula with the following parsing tree: its root is left labelled by $\forall x P(x)$; its child is left labelled by $P(x)$; the leaf is left labelled by x , that is a term.

We define subformulae via parsing trees as in the case of propositional formulae. We will say that a first order (term) formula $(t) \psi$ is a **(subterm) subformula** of $(s) \phi$ if $(t) \psi$ left labels the (unique) parsing tree for $(s) \phi$.

Example 30. For instance, c , x , and $f(x, c)$ are subterms of $f(f(c, x), x)$. Then $\forall x R(x, f(y))$ is a subformula of $\exists y (\forall x R(x, f(y)) \wedge \neg R(y, f(y)))$.

2.1.3 Closed Formulae

Study Section 7.1 of your textbook for intuitions on quantifiers and the notion of free and bound occurrences of variables.

The set of **variables with free occurrences** in a formula ϕ is denoted by $FV(\phi)$. Study its recursive Definition 7.2.6 in pp. 166–167 of your textbook.

Example 31. For instance, x is a variable with free occurrences in $R(x, c)$, $\forall y R(x, y)$, and in $\forall x R(x, c) \wedge P(x)$.

The set of variables of ϕ that have no free occurrences are **bound**.

Example 32. For instance, x is bound in $\forall x R(x, c)$ as well as in $\forall x (R(x, y) \rightarrow P(x))$.

Note 10. Parsing trees allow us to easily check if a variable is bound as follows: whenever the variable occurs in a left label of a leaf n , then the variable must also occur in the right label Qx (with $Q \in \{\forall, \exists\}$) of an ancestor node of n . Study Example 7.2.2 of your textbook.

Now, the interesting formulae.

Definition 9. A σ formula is **closed** (a.k.a., a **sentence**) if all its variables are bound.

Intuitively the closed formulae are translatable into English without ‘meaningless variables’; all occurring variables are quantified over, existentially or universally.

Example 33. Let us consider the signature $\sigma = \langle \in \rangle$ of set theory, whether $\in$ is a binary relation symbol, whose intended semantics is “it belongs to”. Let us consider the following natural language sentence.

There is no set of which every set is a member.

We will formalise this into a closed formula of σ . First we rewrite the sentence in terms of “there exists” and “it belongs to” as follows.

It is not the case that there exists a set so that every set belongs to it.

Then we formalise it as a σ closed formula as follows.

$$\neg(\exists x \forall y (\in x y))$$

The standard writing in maths (no infix notation) would be as follows.

$$\neg(\exists x \forall y (x \in y))$$

Example 34. Let us consider the following set of sentences.

All apples are bad. Some apples are cool. Some apples are bad or cool.

In order to formalise them, we consider σ with the following unary relation symbols: A for “being an apple”; B for “being bad”; C for “being cool”.

The first sentence gets rewritten as

If something is an apple then it is bad.

This is formalised as $\forall x(A(x) \rightarrow B(x))$.

The second sentence gets rewritten as

There exists something that is an apple and is cool.

This is formalised as $\exists x(A(x) \wedge C(x))$.

The third sentence gets rewritten as

If something is an apple then it is bad or cool.

This is formalised as $\forall x(A(x) \rightarrow (B(x) \vee C(x)))$.

For more formalisation examples, see the following subsection.

2.1.4 Examples of First Order Formalisation

Simple sentences

Given that all birds fly, if Tweete is a bird then Tweete flies.

$\forall x (B(x) \rightarrow F(x)) \rightarrow (B(c) \rightarrow F(c))$ with $\sigma = \langle B, F; c \rangle$ where B stands for “to be a bird”, F for “to fly”, and c for “Tweete”.

All natural numbers are odd or even.

Possible formalisations:

- $\forall x (Nat(x) \rightarrow (Odd(x) \vee Even(x)))$ with $\sigma = \langle Nat, Odd, Even \rangle$ where Nat stands for “to be a natural number”, Odd for “to be odd”, $Even$ for “to be even”;
- $\forall x (Nat(x) \rightarrow \exists y(x = prod(2, y) \vee \exists z(z = prod(2, y) \wedge x = succ(z))))$ with $\sigma = \langle Nat; mult, succ; 2 \rangle$ where $mult$ is a two-place function which stands for the multiplication of two naturals and 2 stands for the natural 2;

- $\forall x (Nat(x) \rightarrow \exists y (x = mult(2, y) \vee \exists z (z = mult(2, y) \wedge x = add(z, 1))))$
with $\sigma = \langle Nat; mult, add; 2, 1 \rangle$ where add is a 2-place function which stands for the addition of two naturals and 1 stands for the natural 1.

There exists an integer that is also a natural number.

$\exists x (Z(x) \wedge Nat(x))$ with $\sigma = \langle Nat, Z \rangle$ where Nat is as above and Z is a unary relation symbol which stands for “to be an integer”.

A dudu flies.

If we interpret a as *a certain* then a formalisation is $\exists x (D(x) \wedge F(x))$ with $\sigma = \langle D, F \rangle$ where D stands for “to be a dudu” and F for “to fly”.

If we interpret a as *all* then a formalisation is $\forall x (D(x) \rightarrow F(x))$ with σ as above.

If there is a penguin that flies then all penguins fly.

$\exists x (P(x) \wedge F(x)) \rightarrow \forall x (P(x) \rightarrow F(x))$ with $\sigma = \langle P, F \rangle$ where P stands for “to be a penguin” and F for “to fly”.

Everybody has somebody to love.

$\forall x \exists y L(x, y)$ with $\sigma = \langle L \rangle$ where L , a binary relation symbol, stands for “to love”.

Block world

Consider the signature σ with the following symbols: yellowblock, redblock, and blueblock are the constant symbols; L, O are the binary relation symbols. Use σ to formalise the following set of sentences.

- *The blue block is on top of the yellow block.*
- *The yellow block is to the left of the red block.*
- *Nothing is on top of something and to the left of the red block.*

The resulting set T is as follows:

$$T = \left\{ \begin{array}{l} O(\text{blueblock}, \text{yellowblock}), \\ L(\text{yellowblock}, \text{redblock}), \\ \forall x \neg (\exists y O(x, y) \wedge L(x, \text{redblock})) \end{array} \right\}.$$

Program properties

Write a first-order formula expressing that an array of a given size, say 3, is sorted.

A closed formula expressing this is

$$\forall i \left(\left((0 = i \vee c < i) \wedge i < 3 \right) \rightarrow \left(arr(i) < arr(s(i)) \right) \right)$$

with $\sigma = \langle <; s, arr; 3, 0 \rangle$ where 0 is a constant symbol which stands for 0, 3 is a constant symbol which stands for 3 (the size of the array), s is a unary function symbol and $s(i)$ stands for “successor of i ”, arr is a unary function symbol and $arr(i)$ stands for “ i -th element of the array”, $<$ is a binary relation symbol which stands for the standard linear order over naturals (we wrote $s < t$ instead of $<(s, t)$ for keeping the formula readable).

Graphs

Consider the graph below and the first-order signature σ with only a binary relation symbol R . Formalise the following properties of the graph as σ closed formulae: (i) $(1, 1)$ and $(2, 2)$ are edges of the graph; (ii) $(2, 1)$ is not an edge of the graph.



Note that (i) states that the graph relation is reflexive, which is formalised with the σ formula $\forall x R(x, x)$. Then (ii) can only be formalised as $\exists x \exists y \neg R(x, y)$ with R (because σ has only the binary relation symbol R).

Thereby $\Sigma = \{\forall x R(x, x), \exists x \exists y \neg R(x, y)\}$.

Barber story revisited

Reconsider the barber story with some variants as below.

Suppose that an island is inhabited by at least two persons, Angelo and Roberto. If Angelo shaves an inhabitant then this shaves Roberto; moreover, if Roberto shaves an inhabitant then this does not shave Angelo. Each inhabitant shaves himself.

Formalise the essential facts using a first-order language.

Let $S(x, y)$ stand for “ x shaves y ”, where S is a binary relation symbol. The signature also needs to have a constant for Angelo, say a , and one for Roberto, say r . So, take as signature $\sigma = \langle S; a, r \rangle$. Then Σ is the following set of formulae

$$\left\{ \begin{array}{l} \forall x \ (S(a, x) \rightarrow S(x, r)), \\ \forall x \ (S(r, x) \rightarrow \neg S(x, a)), \\ \forall x \ S(x, x) \end{array} \right\} \quad \begin{array}{l} (1) \\ (2) \\ (3) \end{array}$$

where: (1) stands for “if Angelo shaves an inhabitant then this shaves Roberto”;
 (2) stands for “if Roberto shaves an inhabitant then this does not shave Angelo”;
 (3) stands for “each inhabitant shaves himself”.

2.2 Semantics

2.2.1 Interpretations

It is more complicated to give meaning to a formula of first-order logic than to propositional formulae. We must say how we are interpreting the constant, function and relation symbols with respect to a discourse domain. This is taken care by an interpretation, which assigns to each symbol its ‘meaning’ in the domain. More precisely, a constant symbol will be interpreted as a specific object (element) of the domain, a function symbol will be interpreted as a function over the domain, and a relation symbol will be interpreted as a relation over the domain.

Definition 10. A σ -interpretation¹ is a tuple $\mathcal{I} = (D, \mathcal{C}, \mathcal{F}, \mathcal{R})$ where:

- D is a non-empty set, the **domain**;
- $\mathcal{C}$, $\mathcal{F}$ and $\mathcal{R}$ are sets so defined:
 - for each constant symbol c of σ , $\mathcal{C}$ has precisely an element $c^{\mathcal{I}}$ of D ;
 - for each function symbol f of σ , $\mathcal{F}$ has precisely an n -ary function $f^{\mathcal{I}} : D^n \mapsto D$;
 - for each relation symbol R of σ , $\mathcal{R}$ has precisely an n -ary relation $R^{\mathcal{I}} \subseteq D^n$;
 - $\mathcal{C}$, $\mathcal{F}$ and $\mathcal{R}$ contain nothing else.

If $\mathcal{C}$, $\mathcal{F}$, or $\mathcal{R}$ is empty, we simply skip it.

Note that in your textbook, σ -interpretations are referred to as σ -structures (cfr. Definition 5.5.2). In Section 5.5 of [Chiswell and Hodges, 2007], study Examples 5.5.1 for the signature of arithmetics, σ_{arithm} , and 5.5.4 for the signature of directed graphs, σ_{digraph} .

¹They are referred as σ -structures in [Chiswell and Hodges, 2007].

Example 35. Consider the signature σ with the following symbols: constant symbols *yellowblock*, *blueblock*; one binary relation symbol, *OnTopOf*.

Then $\mathcal{J} = (D, \mathcal{C}, \mathcal{R})$ is a σ -interpretation with

- domain D equal to $\{\text{yellowblock}, \text{blueblock}\}$,
- $\mathcal{C}$ equal to D ,
- $\mathcal{R}$ has precisely the following binary relation:

$$\text{OnTopOf}^{\mathcal{J}} = \{(\text{blueblock}, \text{yellowblock})\}.$$

Note 11. Note that, according to the above definition, there may be more elements in D than constant symbols in the language of σ —as well as relations and functions. In case so, we will free to expand σ with new constant symbols (for $\mathcal{J}$), so that, in the end, for each $\mathbf{d} \in D$ there is at least one constant symbol c with $c^{\mathcal{J}} = \mathbf{d}$ in the expanded signature. The new constant symbols are usually referred to as **witnesses**. With an abuse of notation, we will denote such an expansion of σ with σ again.

In these notes, d will denote *some* constant symbol of the expanded signature so that $d^{\mathcal{J}} = \mathbf{d}$. Such notation is justified by the fact that, for the satisfaction of a closed formula, it does not really matter if $\mathbf{d}$ interprets in $\mathcal{J}$ a constant symbol c and a different constant symbol e , as we will see in a short while.

2.2.2 Substitutions

As in the case of propositional logic, we define formulae obtained by substitutions. The interesting substitutions are those of a constant symbol c for a variable x , thereby assigning to x a value ($c^{\mathcal{J}}$) in the interpretation domain. However, we have to proceed with care: not all substitutions are allowed, e.g., if we substituted c everywhere in $\forall x P(x)$ we would obtain an expression $\forall c P(c)$ that is not a formula, besides contradicting the interpretation of c in $\mathcal{J}$ as a specific element of D . So, we should only allow substitutions of *free* variables for constant symbols.

Next, we formally define the notion of substitution as follows: first we define it on variables; then we uniquely extend it to terms (thanks to the uniqueness parsing of terms). Starting from this, we define the notion of substitution for atomic formulae; finally, we uniquely extend it to all formulae (thanks to the uniqueness parsing of formulae).

Definition 11. Let $\mathcal{J}$ be a σ -interpretation, x a variable free in ϕ of $\text{FOL}(\sigma)$, t a term of σ and c a constant symbol of σ (possibly expanded with constant symbols

for $\mathcal{J}$, see above). A **substitution** of c for x , written as x/c , is mapping from terms and formulae to expressions, defined as follows.

The result of the substitution of c for x in a term t , written as $t(x/c)$, is:

- c if t is x ; t if t is y (no substitution);
- t if t is some constant symbol (no substitution);
- $f(t_1(x/c), \dots, t_n(x/c))$ if t is $f(t_1, \dots, t_n)$.

The result of the substitution of c for x in a formula ϕ , written as $\phi(x/c)$, is:

- $R(t_1(x/c), \dots, t_n(x/c))$ if ϕ is $R(t_1, \dots, t_n)$;
- $(\neg\psi(x/c))$ if ϕ is $(\neg\psi)$;
- $(\chi(x/c) \square \psi(x/c))$ if ϕ is $(\chi \square \psi)$, for every $\square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$;
- $\forall y \psi(x/c)$ if ϕ is $\forall y \psi$ with y different than x ; ϕ if ϕ is $\forall x \psi$ (no substitution).
- $\exists y \psi(x/c)$ if ϕ is $\exists y \psi$ with y different than x ; ϕ if ϕ is $\exists x \psi$ (no substitution).

Example 36. Consider the term x and the constant symbol c . Then $x(x/c)$ is x , and $x(y/c)$ is x .

Consider the formula $R(x, y)$ and the constant symbol c . Then $R(x, y)(x/c)$ is $R(c, y)$, and $R(c, y)(y/c')$ is $R(c, c')$.

Consider the formula $\forall x P(x) \wedge P(x)$ and the constant symbol c . Then the formula $(\forall x P(x) \wedge P(x))(x/c)$ is $\forall x P(x) \wedge P(c)$.

In this manner, substitutions in terms give terms, and substitutions in formulae give formulae. In particular, substitutions of x in formulae in which only x may be free give closed formulae. Moreover, if x is different than y , $\phi(x/c)(y/c')$ (obtained by first applying x/c to ϕ , and subsequently y/c' to $\phi(x/c)$) is (syntactically) equal to $\phi(y/c')(x/c)$ (obtained by first applying y/c' to ϕ , and subsequently x/c to $\phi(y/c')$). Prove them, by simply spelling out the above definitions first for terms with at most x free, and then for formulae with at most x free.

Fact 1. If ϕ is a σ -formula in which at most the variable x is free, then $\phi(x/c)$ is a closed formula for any constant symbol c .

Fact 2. If ϕ is a σ -formula, then $\phi(x/c)(y/c')$ is equal to $\phi(y/c')(x/c)$ for any two distinct variables x and y , and any two constant symbols c and c' .

2.2.3 Satisfiable and Valid Formulae

We have now all the tools for defining when an interpretation satisfies a closed formula². We start by defining the notion of first-order interpretation of closed terms in a σ -interpretation $\mathcal{J}$, and then the notion of satisfaction in $\mathcal{J}$ for all closed formulae starting with atomic closed formulae (thanks to the uniqueness of parsing trees for formulae, and Fact 1).

The **interpretation of the closed term t in $\mathcal{J}$** , written as $t^{\mathcal{J}}$, is defined as follows:

- if t is a constant symbol c , then $t^{\mathcal{J}}$ is $c^{\mathcal{J}}$;
- if t is of the form $f(t_1, \dots, t_n)$ (where $t_1, \dots, t_n$ are closed terms) then $t^{\mathcal{J}}$ is $f^{\mathcal{J}}(t_1^{\mathcal{J}}, \dots, t_n^{\mathcal{J}})$.

Now, let ϕ be a *closed* σ -formula. We will write $\mathcal{J} \models \phi$ if one of the following cases holds, $\mathcal{J} \not\models \phi$ otherwise:

- in case ϕ is an atomic closed formula
 - of the form $R(t_1, \dots, t_n)$, then $(t_1^{\mathcal{J}}, \dots, t_n^{\mathcal{J}}) \in R^{\mathcal{J}}$,
 - of the form $t_1 = t_n$, then $t_1^{\mathcal{J}} = t_n^{\mathcal{J}}$;
- in case ϕ is a closed formula of the form $\neg\psi$ then $\mathcal{J} \not\models \psi$;
- in case ϕ is a closed formula of the form $\chi \wedge \psi$ then $\mathcal{J} \models \chi$ and $\mathcal{J} \models \psi$;
- in case ϕ is a closed formula of the form $\chi \vee \psi$ then $\mathcal{J} \models \chi$ or $\mathcal{J} \models \psi$;
- in case ϕ is a closed formula of the form $\chi \rightarrow \psi$ then $\mathcal{J} \not\models \chi$ or $\mathcal{J} \models \psi$;
- in case ϕ is a closed formula of the form $\chi \leftrightarrow \psi$ then $\mathcal{J} \models \chi$ iff $\mathcal{J} \models \psi$;
- in case ϕ is a closed formula of the form $\forall x\psi$ then $\mathcal{J} \models \psi(x/d)$ for all $d \in D$;
- in case ϕ is a closed formula of the form $\exists x\psi$ then $\mathcal{J} \models \psi(x/d)$ for some $d \in D$.

²At this point, a standard logic book will give a more general definition of satisfaction for formulae, not necessarily closed—the Tarski definition. To this end, the book will first introduce assignments for variables into the domain of a σ -interpretation, something which we skip in our notes. Our simplified (and simplistic) definition is based on the one in Chapter 9 of [Barwise, J. and Etchemendy, J., 1991] and Chapter 5 of [Chiswell and Hodges, 2007], and is geared towards the first-order tableau calculus considered in this course.

Remember that d denotes some constant symbol of σ with $d^J = \mathbf{d}$, see Note 11. Could it then be that we have two distinct constant symbols, say c and e , with $c^J = \mathbf{d} = e^J$, and so that $J \models \psi(x/c)$ but $J \not\models \psi(x/e)$? If so, then the last two clauses of the above definition would not make sense. . . Luckily, this is not the case due to the following theorem, the proof of which requires first a lemma for terms.

Lemma 5. Consider a σ -interpretation J with domain D . Let t be a term of σ in which at most x is free. Let c and e be two distinct constant symbols of σ , with $c^J = e^J$. Then $t(x/c)^J$ is equal to $t(x/e)^J$.

Proof. If x is not free in t , the thesis follows immediately from the definition of substitution: $t(x/c)$ and $t(x/e)$ are (syntactically) equal to t .

So, let x be the only free variable of t . Then $t(x/c)$ is (syntactically) different than $t(x/e)$, but both are closed (see Fact 1). We prove that the elements $t(x/c)^J$ and $t(x/e)^J$ of D are equal, working by induction on t with x free.

If t is x then $t(x/c)$ is c and $t(x/e)$ is e . The result follows from the hypothesis $c^J = e^J$. (We do not consider the cases of t equal to a variable different than x or a constant symbol because in both cases x would not be free in t .)

If t is $f(t_1, \dots, t_n)$ we apply the induction hypothesis to all t_i , $i = 1, \dots, n$ and conclude that $t_i(x/c)^J$ is equal to $t_i(x/e)^J$ for all $i = 1, \dots, n$. Now, by definition of substitution and interpretation of a closed term, $t(x/c)^J$ is equal to $f^J(t_1(x/c)^J, \dots, t_n(x/c)^J)$. The latter, by induction hypothesis, is equal to $f^J(t_1(x/e)^J, \dots, t_n(x/e)^J)$, which is $t(x/e)^J$ by definition of substitution and interpretation of a closed term. Therefore $t(x/c)^J$ and $t(x/e)^J$ are equal. $\square$

Theorem 20. Consider a σ -interpretation J with domain D . Let ϕ be a formula of σ in which at most x is free. Consider $\mathbf{d} \in D$. If $J \models \phi(x/c)$ for *some* constant symbol c with $c^J = \mathbf{d}$ then $J \models \phi(x/e)$ for *every* constant symbol e with $e^J = \mathbf{d}$.

Proof. If x is not free in ϕ , the thesis follows immediately from the definition of substitution: $\phi(x/c)$ and $\phi(x/e)$ are (syntactically) equal to ϕ .

So, let x be the only free variable of ϕ . Then $\phi(x/c)$ is (syntactically) different than $\phi(x/e)$, but both are closed (see Fact 1). We prove that $J \models \phi(x/c)$ iff $J \models \phi(x/e)$ by induction on ϕ .

If ϕ is atomic of the form $R(t_1, \dots, t_n)$ then $\phi(x/c)$ is $R(t_1(x/c), \dots, t_n(x/c))$, and $\phi(x/e)$ is $R(t_1(x/e), \dots, t_n(x/e))$. Then, $J \models \phi(x/c)$ iff $(t_1(x/c)^J, \dots, t_n(x/c)^J) \in R^J$. Similarly, $J \models \phi(x/e)$ iff $(t_1(x/e)^J, \dots, t_n(x/e)^J) \in R^J$. Above, we proved that the element $t_i(x/c)^J$ of D is equal to the element $t_i(x/e)^J$ of D , for all $i = 1 \dots n$. So $(t_1(x/c)^J, \dots, t_n(x/c)^J) \in R^J$ iff $(t_1(x/e)^J, \dots, t_n(x/e)^J) \in R^J$. Therefore $J \models \phi(x/c)$ iff $J \models \phi(x/e)$.

The propositional cases are treated as usual. Let us see the case of ϕ of the form $\forall y\psi$, with y different than x (it cannot be of the form $\forall x\psi$, else x would not be free in ϕ).

Assume that $\mathcal{J} \models \forall y\psi(x/c)$ holds. This is the case iff, for all $\mathbf{d}' \in D$ and *some* c' in the signature with $c'^{\mathcal{J}} = \mathbf{d}'$, we have that $\mathcal{J} \models \psi(x/c)(y/c')$. Take an arbitrary $\mathbf{d}'$ of D . By induction hypothesis, we have that for *every* c' with $c'^{\mathcal{J}} = \mathbf{d}'$, we have that $\mathcal{J} \models \psi(x/c)(y/c')$. Now, $\psi(x/c)(y/c') = \psi(y/c')(x/c)$ since x and y are different (see Fact 2). We conclude that, for every c' in σ , $\mathcal{J} \models \psi(y/c')(x/c)$. By induction hypothesis, we get that $\mathcal{J} \models \psi(y/c')(x/e)$, for every c' in σ . Therefore $\mathcal{J} \models \forall y\psi(x/e)$. The vice-versa is proved similarly. $\square$

If $\mathcal{J} \models \phi$ we will say that the σ -**interpretation** $\mathcal{J}$ **satisfies**³ (or is a **model** of) the closed σ -formula ϕ . Otherwise, if $\mathcal{J} \not\models \phi$, we will say that the σ -interpretation $\mathcal{J}$ **falsifies** (or is a **counter-model** of) the closed σ -formula ϕ .

More in general, we will say that a *closed formula* ϕ of σ is

- **satisfiable** if there exists a σ -model $\mathcal{J}$ so that $\mathcal{J} \models \phi$;
- **falsifiable** if there exists a σ -model $\mathcal{J}$ so that $\mathcal{J} \not\models \phi$;
- **valid** if, for all σ -models $\mathcal{J}$, we have that $\mathcal{J} \models \phi$;
- **unsatisfiable** if, for all σ -models $\mathcal{J}$, we have that $\mathcal{J} \not\models \phi$.

Example 37. Consider the signature σ as follows: *yellowblock*, and *blueblock* are the constant symbols; L is the binary relation symbols. The closed σ -formula $\phi = \text{OnTopOf}(\text{blueblock}, \text{yellowblock})$ formalises the sentence “The blue block is on top of the yellow block”.

The following σ -interpretation $\mathcal{J}$, depicted as the left world in Figure 2.3,

- with domain $D = \{\text{yellowblock}, \text{blueblock}\}$,
- set of constants equal to D ,
- and a binary relation, namely, $\text{OnTopOf}^{\mathcal{J}} = \{(\text{blueblock}, \text{yellowblock})\}$,

is a model of ϕ .

The following σ -interpretation $\mathcal{J}'$, depicted as the right world in Figure 2.3,

³We could easily extend the above definition to any formula, even non closed. Let $x_1, \dots, x_n$ be all the free variables of ϕ . The **universal closure** of ϕ is the formula $\forall x_1 \dots \forall x_n \phi$. The **existential closure** of ϕ is the formula $\exists x_1 \dots \exists x_n \phi$. Next, we will say that $\mathcal{J}$ **validates** ϕ if $\mathcal{J} \models \forall x_1 \dots \forall x_n \phi$. We will say that $\mathcal{J}$ **satisfies** ϕ if $\mathcal{J} \models \exists x_1 \dots \exists x_n \phi$. As we will not be using such notions, we leave them in a note for the curious student.

- with domain $D' = \{\text{yellowblock}, \text{blueblock}\}$,
- set of constants equal to D' ,
- and a binary relation, namely, $\text{OnTopOf}^{\mathcal{J}'}$, which is empty (no element is related to another by $\text{OnTopOf}^{\mathcal{J}'}$),

is a counter-model of ϕ . Therefore ϕ is satisfiable and falsifiable, and hence it is neither valid nor unsatisfiable.

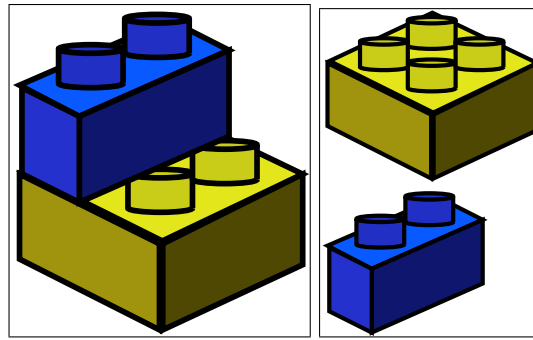


Figure 2.3: Two block worlds.

Example 38. The formula $\forall xP(x) \vee \neg\forall xP(x)$ is valid—it is a propositional valid formula, if we regard $\forall xP(x)$ as a propositional symbol. In fact, for all $\mathcal{J}$, we have that $\mathcal{J} \models \forall xP(x)$ or not, that is, $\mathcal{J} \models \forall xP(x) \vee \neg\forall xP(x)$.

Proving that $\forall xP(x) \vee \exists x\neg P(x)$ is valid requires more than a propositional semantic argument. We have to prove that, for all $\mathcal{J}$, $\mathcal{J} \models \forall xP(x)$ or $\mathcal{J} \models \exists x\neg P(x)$. We reason on the disjuncts separately. The former holds iff $\mathbf{d}$ belongs to $P^{\mathcal{J}}$ for all $\mathbf{d}$ in the domain of $\mathcal{J}$. The latter holds iff there exists $\mathbf{d}$ in the domain of $\mathcal{J}$ that does not belong to $P^{\mathcal{J}}$. Summing up, $\forall xP(x) \vee \neg\exists x\neg P(x)$ is equivalent to saying that all $\mathbf{d}$ of the domain of $\mathcal{J}$ belongs to $P^{\mathcal{J}}$ or there exists $\mathbf{d}$ in the domain of $\mathcal{J}$ that does not belong to $P^{\mathcal{J}}$, which is true.

The negation of any of the above two (valid) formulae is unsatisfiable—if all interpretations are models of ϕ , there is no interpretation that is a model of $(\neg\phi)$.

2.2.4 Set Satisfiability

As in the propositional case, the notion of satisfaction can be extended to sets of closed formulae. In case of first order logic, a set of closed (first order) formulae deserves a special name: it is a **theory**. You encountered theories in algebra: the

set of closed formulae defining groups is the theory of groups, similarly, the set T with formulae

$$\begin{aligned}\forall x \forall y \quad & (R(x, y) \vee R(y, x)) \\ \forall x \forall y \quad & (\neg(R(x, y) \wedge R(y, x))) \\ \forall x \forall y \forall z \quad & (R(x, y) \wedge R(y, z) \rightarrow R(x, z))\end{aligned}$$

is the theory of total orderings, i.e., if $\mathcal{J} = (D, \{R^{\mathcal{J}}\})$ is a model of *all* the formula of T then $R^{\mathcal{J}}$ is a total order (and vice-versa).

The latter example gives us the key for formally defining when an interpretation satisfies a theory.

Definition 12. Let T be a σ -theory.

If $\mathcal{J}$ is σ -interpretation, we will say that $\mathcal{J}$ **satisfies** T , or is a **model** of T , written as $\mathcal{J} \models T$, if $\mathcal{J}$ satisfies every formula of T ; it **falsifies** T , or is a **counter-model** of T , written as $\mathcal{J} \not\models T$, if it does not satisfy at least one formula of T .

More in general, we will say that T is:

- **satisfiable** if there exists a σ -model $\mathcal{J}$ so that $\mathcal{J} \models T$;
- **falsifiable** if there exists a σ -model $\mathcal{J}$ so that $\mathcal{J} \not\models T$;
- **valid** if, for all σ -models $\mathcal{J}$, we have that $\mathcal{J} \models T$;
- **unsatisfiable** if, for all σ -models $\mathcal{J}$, we have that $\mathcal{J} \not\models T$.

As in the case of propositional logic, we can prove the following result.

Theorem 21. Let $T = \{\phi_1, \dots, \phi_n\}$ be a finite σ -theory. T is satisfiable iff the (closed) formula $\bigwedge_{i=1}^n \phi_i$ is satisfiable.

Proof. It follows from the definition of $\mathcal{J} \models \chi \wedge \psi$. □

Therefore, similarly to the case of propositional logic, we can reduce the problem of the satisfiability of a finite theory to the problem of the satisfiability of the conjunction of the formulae of the theory, and vice-versa since the satisfiability of a closed formula is a special case of satisfiability of a theory (containing only the closed formula). However, contrary to propositional logic, we have no algorithm, so far, for deciding the one nor the other problem.

Example 39. Consider the block world example in Subsubsection 2.1.4, with signature σ as follows: *yellowblock*, *redblock*, and *blueblock* are the constant symbols; L , O are the binary relation symbols. We used σ to formalise the following set of sentences.

- The blue block is on top of the yellow block.
- The yellow block is to the left of the red block.
- Nothing is on top of something and to the left of the red block.

The resulting set T , which is a theory, is as follows:

$$T = \left\{ \begin{array}{l} OnTopOf(blueblock, yellowblock), \\ LeftOf(yellowblock, redblock), \\ \forall x \neg (\exists y O(x, y) \wedge L(x, redblock)) \end{array} \right\}.$$

The following σ -interpretation $\mathcal{J}$, depicted as the left world in Figure 2.4,

- with domain $D = \{\text{yellowblock}, \text{blueblock}, \text{redblock}\}$,
- set of constants equal to D ,
- and two binary relations, namely, $OnTopOf^{\mathcal{J}} = \{(\text{blueblock}, \text{yellowblock})\}$ and $LeftOf^{\mathcal{J}} = \{(\text{yellowblock}, \text{redblock})\}$,

is a model of T .

The following σ -interpretation $\mathcal{E}$, depicted as the right world in Figure 2.4,

- with domain $D' = \{\text{yellowblock}, \text{blueblock}, \text{redblock}\}$,
- set of constants equal to D' ,
- and two binary relations, namely, $OnTopOf^{\mathcal{J}'} = \{(\text{blueblock}, \text{yellowblock})\}$, and $LeftOf^{\mathcal{J}'} = \{(\text{redblock}, \text{yellowblock})\}$,

is a counter-model of T , as it does not satisfy the closed formula

$$LeftOf(yellowblock, redblock).$$

Therefore T is satisfiable and falsifiable, and hence it is neither valid nor unsatisfiable.

Example 40. Consider the first-order language $\sigma := \langle R \rangle$ where R is a binary relation symbol, and the σ -theory $T' = \{\forall x R(x, x), \forall x \neg R(x, x)\}$. We prove that T' is unsatisfiable by using the definition of set unsatisfiability. We have to check the following, for all σ -interpretations $\mathcal{J} = (D, R^{\mathcal{J}})$, with $R^{\mathcal{J}} \subseteq D \times D$: if $\mathcal{J} \models T'$ then $\mathcal{J} \not\models \forall x \neg R(x, x)$. Take an arbitrary $\mathcal{J}$. We have to prove that if $\mathcal{J} \models T'$ then $\mathcal{J} \not\models \forall x \neg R(x, x)$. Now $\mathcal{J} \models T'$ means that $(a, a) \in R^{\mathcal{J}}$ for all $a \in D$. Since D is not empty, there exists $a \in D$. So, in particular, $(a, a) \in R^{\mathcal{J}}$. Hence, $\mathcal{J} \models R(a, a)$, that is, $\mathcal{J} \not\models \neg R(a, a)$. This means that $\mathcal{J} \not\models \forall x \neg R(x, x)$.

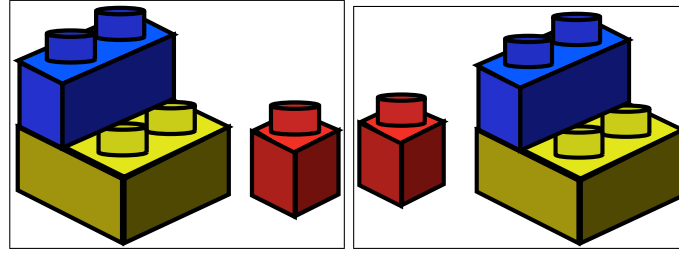


Figure 2.4: Two block worlds.

2.2.5 Entailment and Logical Equivalence

As in the case of propositional logic, we define the notion of entailment between a theory and a closed formula. We say that the σ -theory T **entails** the closed σ -formula ϕ , and write $T \models \phi$, if

for all σ -interpretations $\mathcal{J}$, if $\mathcal{J} \models \Theta$ then $\mathcal{J} \models \phi$.

Having $\neg$ in the language, entailment and set (un)satisfiability are complementary problems in the following sense.

Lemma 6. Let T be a σ -theory and ϕ be a closed σ -formula. $T \models \phi$ iff $T \cup \{(\neg\phi)\}$ is unsatisfiable.

Proof. As in the case of propositional logic. □

In case $T = \{\phi_1, \dots, \phi_n\}$, thanks to the above Lemma 6, we can verify whether $T \models \phi$ or not by verifying the unsatisfiability of $T \cup \{(\neg\phi)\}$. This can be done by checking the unsatisfiability of $\bigwedge_{i=1}^n \phi_i \wedge (\neg\phi)$ (see Theorem 21).

Theorem 22. Let $\{\phi_1, \dots, \phi_n\} \cup \{\phi\}$ be a σ -theory and ϕ a closed σ -formula. We have that $\{\phi_1, \dots, \phi_n\} \models \phi$ iff $\bigwedge_{i=1}^n \phi_i \wedge (\neg\phi)$ is an unsatisfiable theory. □

Therefore, similarly to the case of propositional logic, we can reduce the problem of the entailment of a closed formula from a finite theory to that of the unsatisfiability of a finite theory, and vice-versa. However, contrary to propositional logic, we have no algorithm, so far, for deciding the one nor the other problem.

Example 41. Consider the first-order language $\sigma := \langle R \rangle$ where R is a binary relation symbol, and the σ -theory $T = \{\forall x \forall y R(x, y)\}$. Now, we prove that $T \models \exists x R(x, x)$ using the definition of entailment. We have to check that, for all σ -interpretations $\mathcal{J} = (D, R^{\mathcal{J}})$, with $R^{\mathcal{J}} \subseteq D \times D$, if $\mathcal{J} \models T$ then $\mathcal{J} \models \exists x R(x, x)$. Take an arbitrary $\mathcal{J}$. We have to prove that if $\mathcal{J} \models T$ then $\mathcal{J} \models \exists x R(x, x)$. Now,

$\mathcal{J} \models T$ if $\mathcal{J} \models \forall x \forall y R(x, y)$; by definition, this is true if $(\mathbf{a}, \mathbf{b}) \in R^{\mathcal{J}}$ for *every* $\mathbf{a}$, and for *every* $\mathbf{b} \in D$. D is not empty and say that $\mathbf{a} \in D$. So in particular, $(\mathbf{a}, \mathbf{a}) \in R^{\mathcal{J}}$. Therefore there exists $\mathbf{a}$ so that $\mathcal{J} \models R(\mathbf{a}, \mathbf{a})$, and hence $\mathcal{J} \models \exists x R(x, x)$.

As with propositional logic, in case $T = \{\psi\}$, that is, T contains only one formula, then we drop brackets and write $\psi \models \phi$. In particular, two formulae ψ and ϕ are **logically equivalent**, written as $\psi \equiv \phi$, if $\psi \models \phi$ and $\phi \models \psi$. In other words, two formula are semantically equivalent if, independently of their syntactic writing, they carry the same semantic information.

Theorem 23. $\phi \equiv \psi$ iff $(\phi \leftrightarrow \psi)$ is valid.

Proof. As in the propositional case, the result follows from the definitions of $\equiv$ and $\mathcal{J} \models \chi \leftrightarrow \psi$. $\square$

Example 42. The following are all equivalent closed formulae—prove it!

$$\begin{aligned}
 \forall x \phi \wedge \psi &\equiv \forall x(\phi \wedge \psi) \text{ if } x \text{ is not free in } \psi \\
 \forall x \phi \vee \psi &\equiv \forall x(\phi \vee \psi) \text{ if } x \text{ is not free in } \psi \\
 \exists x \phi \wedge \psi &\equiv \exists x(\phi \wedge \psi) \text{ if } x \text{ is not free in } \psi \\
 \exists x \phi \vee \psi &\equiv \exists x(\phi \vee \psi) \text{ if } x \text{ is not free in } \psi \\
 \forall x \phi \wedge \forall x \psi &\equiv \forall x(\phi \wedge \psi) \\
 \exists x \phi \vee \exists x \psi &\equiv \exists x(\phi \vee \psi) \\
 \neg \forall x \phi &\equiv \exists x \neg \phi \\
 \neg \exists x \phi &\equiv \forall x \neg \phi
 \end{aligned}$$

Hereby, we prove the top one, and the bottom two.

Assume that ψ is closed. Now, $\mathcal{J} \models \forall x \phi \wedge \psi$ iff $\mathcal{J} \models \forall x \phi$ and $\mathcal{J} \models \psi$. Let us analyse $\mathcal{J} \models \forall x \phi$ and $\mathcal{J} \models \psi$, separately. The former is equivalent to saying that $\mathcal{J} \models \psi(x/a)$ for all $\mathbf{a} \in D$. Since x is not free in ψ , $\psi(x/a)$ is equal to ψ , and hence, trivially, $\mathcal{J} \models \psi$ is the same as $\mathcal{J} \models \psi(x/a)$. In brief, $\mathcal{J} \models \forall x \phi$ and $\mathcal{J} \models \psi$ iff $\mathcal{J} \models \phi(x/a)$ and $\mathcal{J} \models \psi(x/a)$ for all $\mathbf{a} \in D$. This is equivalent to saying that $\mathcal{J} \models \forall x(\phi \wedge \psi)$.

Now, $\mathcal{J} \models \neg \forall x \phi$ iff $\mathcal{J} \not\models \forall x \phi$, that is, iff it is false that $\mathcal{J} \models \phi(x/a)$ for *all* $\mathbf{a} \in D$. This is equivalent to saying that there exists some $\mathbf{a} \in D$ so that $\mathcal{J} \models \neg \phi(x/a)$. This is equivalent to saying that $\mathcal{J} \models \exists x \neg \phi$.

Last, $\mathcal{J} \models \neg \exists x \phi$ iff $\mathcal{J} \not\models \exists x \phi$, that is, iff it is false that $\mathcal{J} \models \phi(x/a)$ for *some* $\mathbf{a} \in D$. This is equivalent to saying that, for every $\mathbf{a} \in D$, we have that $\mathcal{J} \models \neg \phi(x/a)$. This is equivalent to saying that $\mathcal{J} \models \forall x \neg \phi$.

2.3 Proof Theory

As in the case of propositional logic, proof calculi for first order logic can be distinguished in: human oriented calculi like natural deduction (see Section 7.4 of your textbook, as well as the introductory notes for an informal version of natural deduction), complete for entailment; computer oriented calculi like the analytic tableau calculus that is refutation complete, only.

Hereby we focus on the latter, and restrict our attention to closed formulae, as our goal is to prove that a finite theory T entails a closed formula ϕ iff there is a closed tableau stating the unsatisfiability of the theory $T \cup \{\neg\phi\}$.

In the remainder, we will assume that σ has an enumerable set of symbols, no function symbols, nor $=$. Albeit the assumption seems restrictive, it is not: every formula of a language with an n -ary function symbols is equivalent to a formula of a language that replaces the function symbol with an $n+1$ -ary relation symbol, and formulae stating that this relation symbol must behave like a function—curious students can consult Section 2.7.1 of [Bradley and Manna, 2007].

2.3.1 Analytic Tableau Rules

The rules of the tableau calculus for first order logic extend those for propositional logic with four rules, see Table 2.1: two for the universal quantifier; two for the existential quantifier. The intuitive semantic interpretation of the new rules is in the table.

Notice that γ rules have the form

$$\frac{\gamma}{\gamma[x/a]}$$

where a is a constant symbol, whereas δ rules have the form

$$\frac{\delta}{\delta[x/a]}$$

where a will be subject to the strong restraint of being a *new* constant symbol, not occurring in the left labels of the tableau so far constructed.

Why such a restraint? The δ rules formalise the following informal mathematical argument in proofs (we used it as well in our proofs). Suppose that we proved the existence of a certain x with the property $P(x)$, that is, we proved $\exists xP(x)$. In a classroom lecture you probably have heard a version of “let a be something having the property P ”. Then you continue the proof by substituting a for x , and hence

writing $P(a)$. Later on, if you prove the existence of a certain x with another property Q , that is, $\exists xQ(x)$, you cannot substitute again a for x and write $Q(a)$; a names an object with property P , and you cannot be sure that that object has also property Q !

In case we run out of constant symbols because the set $\mathcal{C}$ of constant symbols of σ is finite, the solution is to extend $\mathcal{C}$ with an enumerable set of constant symbol which are ‘uncommitted’ in the tableau so far constructed, and can be used in δ rules. Such new constant symbols are often referred to as **Skolem constants**. The extended signature is denoted by σ^S .

Rule	Rule name	Rule type	Semantic Argument
$\frac{\forall x\psi}{\psi[x/a]}$	$\forall$ -rule with x/a	γ	if $\mathcal{J} \models \forall x\psi$ then $\mathcal{J} \models \psi[x/a]$ for all a in the domain of $\mathcal{J}$
$\frac{(\neg\forall x\psi)}{(\neg\psi[x/a])}$	$\neg\forall$ -rule with x/a	δ	if $\mathcal{J} \models \neg(\forall x\psi)$ then $\mathcal{J} \models (\neg\psi[x/a])$ for some a in the domain of $\mathcal{J}$
$\frac{\exists x\psi}{\psi[x/a]}$	$\exists$ -rule with x/a	δ	if $\mathcal{J} \models \exists x\psi$ then $\mathcal{J} \models \psi[x/a]$ for some a in the domain of $\mathcal{J}$
$\frac{(\neg\exists x\psi)}{(\neg\psi[x/a])}$	$\neg\exists$ -rule with x/a	γ	if $\mathcal{J} \models (\neg\exists x\psi)$ then $\mathcal{J} \models (\neg\psi[x/a])$ for all a in the domain of $\mathcal{J}$

Table 2.1: Tableau rules specific for first order formulae of the form $\forall x\psi$, $(\neg\forall x\psi)$, $\exists x\psi$, and $(\neg\exists x\psi)$.

2.3.2 Tableaux

Prelude. As in the case of propositional logic, the tableau rules are only concerned with complex closed formulae. Closed formulae that are not complex label nodes that are not used in building the tableau. More precisely, a **closed literal**

of a first order language is a closed atomic formula $p(c_1, \dots, c_n)$ or its negation $\neg p(c_1, \dots, c_n)$. Two ground literals are **complementary**, written as l and l^- , if one is of the form $p(c_1, \dots, c_n)$ and the other is of the form $\neg p(c_1, \dots, c_n)$.

A **branch** of a first order tableau is **closed** if a pair of complementary literals left labels the branch, or $\perp$ left labels it, or $\neg\top$ left labels it. Else, it is **open**. Again, a node is **unusable** if its left label is either $\top$ or its negation, $\perp$ or its negation, a closed literal or in square brackets. Otherwise the node is **usable**.

We need to guarantee that each rule is applied fairly, in particular, that infinite applications of the γ rule does not prevent the application of other rules. To this end, the procedure chooses nodes that are as high as possible in the tableau, systematically.

Procedure. The procedure starts with ϕ as left label of the root r of T , without right label.

While there are usable nodes in open branches of T , *choose* a usable node n *as high as possible in the tableau*.

For each open **branch** passing through n , proceed by cases as follows.

Case α -rule or β -rule. Enclose n in square brackets and proceed as in the propositional case.

Case δ -rule. Enclose n in square brackets. If n is left labelled by a δ formula, then enclose n in square brackets and take the first constant symbol a of σ^S that does *not* occur in the left labels of the current *tableau*. Add a child to n . Right label the child with the appropriate “ δ -rule with x/a ”, and left label it with the rule conclusion $\delta[x/a]$.

$$\begin{array}{c} [\delta] n \\ \vdots \\ | \\ \delta[x/a] n_1 \end{array} \quad \delta\text{-rule with } x/a$$

Case γ -rule. If n is left labelled by a γ formula take the first constant symbol of σ^S so that $\gamma[x/a]$ does not already left label **branch**. Extend **branch** with a node right labelled by the appropriate “ γ -rule with x/a ”, and left labelled by the rule conclusion $\gamma[x/a]$.

$$\begin{array}{c} \gamma n \\ \vdots \\ | \\ \gamma[x/a] n_1 \end{array} \quad \gamma\text{-rule with } x/a$$

Not enclosing γ formulae in square brackets ensures that the procedure can still use them, and hence that, *at the limit*, $\gamma[x/a]$ holds (or not) for *all* possible constant symbols.

Example 43. The following is a closed tableau for $\neg(\forall xP(x) \vee \exists x\neg P(x))$. We skip the right labels; students should add them.

$$\begin{array}{c}
 [\neg(\forall xP(x) \vee \exists x\neg P(x))] \\
 | \\
 [\neg\forall xP(x)] \\
 [\neg\exists x\neg P(x)] \\
 | \\
 \neg P(a) \\
 | \\
 [\neg\neg P(a)] \\
 | \\
 P(a) \text{ closed}
 \end{array}$$

Note that the same constant symbol a , introduced by the $\neg\forall$ -rule, can be used by the $\neg\exists$ -rule.

The following is a *wrongly constructed* tableau for $\neg(\forall xP(x) \vee \forall x\neg P(x))$: what is wrong is the re-use of the constant symbol a by the $\neg\forall$ -rule, which would instead require a new constant symbol different than a . With this fix, however, the above procedure will return an infinite open tableau—do it.

$$\begin{array}{c}
 [\neg(\forall xP(x) \vee \forall x\neg P(x))] \\
 | \\
 [\neg\forall xP(x)] \\
 [\neg\forall x\neg P(x)] \\
 | \\
 \neg P(a) \\
 | \\
 [\neg\neg P(a)] \\
 | \\
 P(a) \text{ closed}
 \end{array}$$

The following is another example of an infinite open tableau, constructed according to the above procedure. Consider the closed formula $\forall xP(x)$. The interpretation $\mathcal{I}$ with a single element a_0 and $P^{\mathcal{I}} = \{a_0\}$ is clearly a model of the formula. The above procedure, at the limit, will generate an infinite open branch with set of left labels $\{P(a_n) \mid n \in \mathbb{N}\}$.

However, if there are no γ formulae, the tableau returned by the above procedure is finite. Let us see an example of a finite open tableau for $\exists x(R(x) \vee \exists yP(y))$, built according to the above procedure.

$$\begin{array}{c}
[\exists x(R(x) \vee \exists yP(y))] \\
\downarrow \\
[R(a) \vee \exists yP(y)] \\
\swarrow \quad \searrow \\
R(a) \quad [\exists yP(y)] \\
\quad \downarrow \\
\quad P(a)
\end{array}$$

Note that we can use the same a in both the applications of the $\exists$ -rule because they are in different branches of the tableau.

Note 12. Now, as the above example shows, the tableau built by the procedure may have

- closed branches, which are always finite (that is, with finitely many nodes),
- finite open branches (e.g., in case no γ formula left labels the branch),
- infinite open branches.

The last case means that the first order tableau procedure may not terminate (with input satisfiable closed formulae): in fact, γ rules allow for building infinite branches. $\square$

2.3.3 Satisfiability and Validity

Despite the above tableau procedure may not terminate, the tableau calculus is refutation complete and sound in the following sense.

Soundness is proved as in the propositional case.

Theorem 24 (Soundness).

- If $\mathcal{I} \models \phi$ then any tableau for ϕ has an open branch, finite or infinite, that $\mathcal{I}$ satisfies.
- If a tableau for ϕ has only closed branches then ϕ is unsatisfiable. $\square$

Completeness of the procedure is proved like in the propositional case, that is,

- we show how any open branch for ϕ defines $\mathcal{I}$ so that $\mathcal{I} \models \phi$,
- and this yields (refutation) completeness,

last, we prove (or better: sketch the proof of the fact) that

- a tableau for an unsatisfiable ϕ , built following the above procedure, is necessarily *finite*, that is, the tableau procedure for ϕ unsatisfiable terminates.

Theorem 25 (Completeness).

- An open branch of a tableau for ϕ built with the above procedure defines an interpretation that satisfies ϕ , possibly at the limit.
- If ϕ is unsatisfiable then a tableau for ϕ has only closed finite branches.

Proof.

- Let the domain D of $\mathcal{J}$ be equal to all the constant symbols that occur in left labels of the branch. Now, take all closed literals that left label the branch: if the literal is of the form $P(c_1, \dots, c_n)$ then set $(c_1, \dots, c_n) \in P^{\mathcal{J}}$; if the literal is of the form $\neg P(c_1, \dots, c_n)$ then set $(c_1, \dots, c_n) \notin P^{\mathcal{J}}$. In this manner, $\mathcal{J}$ is well defined (as the branch is open).

Finally, we prove that this $\mathcal{J}$ satisfies the branch by induction on ψ , left labelling the branch, as in the propositional case. The atomic case is covered by the definition of $\mathcal{J}$. If ψ is an α or β formula, the proof goes exactly as in the propositional case. Assume that ψ is a γ formula. Then $\gamma[x/c]$ is in the branch, for every $c \in D$. By induction hypothesis, $\mathcal{J} \models \gamma[x/c]$, for every $c \in D$. Thus $\mathcal{J} \models \gamma$. Finally, assume that ψ is a δ formula. Then $\delta[x/c]$ is in the branch, for some $c \in D$. By induction hypothesis, $\mathcal{J} \models \delta[x/c]$, for some $c \in D$. Thus $\mathcal{J} \models \delta$.

- The first statement yields that a tableau for ϕ has only closed branches, which are finite.

□

Note that, by König lemma, any tree with finite branches and finitely branching must be finite. Completeness yields that, if ϕ is valid, then the tableau for $\neg\phi$ has only closed branches, that are finite. This and König lemma yield that the tableau is finite, and hence that we can use tableaux for verifying that ϕ is valid—but if it is not valid, then $\neg\phi$ is satisfiable, and the tableau procedure may not terminate with input a satisfiable formula, see Example 43.

Theorem 26 (Semidecidability). If ϕ is valid then there is a procedure that terminates and verifies the validity of ϕ .

The procedure consists in building the tableau for $\neg\phi$: since this is unsatisfiable, it has a finite tableau with only closed branches.

Example 44. Verify the validity of $\forall xP(x) \rightarrow \exists xP(x)$ using the tableau procedure.

$$\begin{array}{c}
 [\neg(\forall xP(x) \rightarrow \exists xP(x))] \\
 | \\
 [\forall xP(x)] \\
 [\neg\exists xP(x)] \\
 | \\
 P(a) \\
 | \\
 \neg P(a) \\
 \text{closed}
 \end{array}$$

The tableau is closed, and hence the formula is valid.

Note 13. The last tableau of Example 43 shows that, even if the input formula is satisfiable in a finite model, the tableau procedure unnecessarily builds an infinite open branch, at the limit—that is, an infinite model. However, in some cases, this can be remedied by (1) scheduling the application of the rules in a certain order, and (2) posing restraints on the constant symbols used in γ rules. The optimised procedure is sketched as follows.

Expand the branch by first applying, in the order, the α -rules, β -rules and δ -rules (applicable only finitely many times). If the branch is not closed, continue. Let A be the set of constant symbols occurring in some left label *of the branch*. If A is empty (e.g., there is no δ formula in the branch), add the first constant symbol of σ^S to A . Then apply all γ -rules with all constant symbols of A . Expand the branch with all the conclusions that do not already occur as left labels in the branch. If the branch is not closed and not expanded (that is, no new conclusions were added), then terminate the procedure, and check that the following structure $\mathcal{J}$, built as in the proof of Theorem 25, is a model of ϕ : the domain of $\mathcal{J}$ is equal to the set of constant symbols occurring in left labels of the branch; for each n -ary relation symbol P in the branch, $(c_1, \dots, c_n) \in P^{\mathcal{J}}$ iff $P(c_1, \dots, c_n)$ occurs as left label of the branch.

The sketched procedure can be refined into an algorithm for deciding on the satisfiability of a formula when this is *satisfiable in a finite model*, see [Peltier, 2003].

However, there is no way of obtaining a decision procedure (that is, sound, complete and terminating) for any first-order formula in view of the following landmark result by Church (but equivalent to other landmark results achieved by different logicians, e.g., Gödel, Matiyasevich, Rosser, Turing⁴):

Church Undecidability Theorem: there is no algorithm that decides on the validity, or not, of an arbitrary formula of an arbitrary

⁴If you are interested, see Chapter 8 in your textbook [Chiswell and Hodges, 2007]. However, this is not part of the course.

first-order language.

In brief, unlike in propositional logic, the validity problem in first-order logic is only semidecidable (Theorem 26), and not decidable (Church Undecidability Theorem). As [Fitting, 1997] puts it, “there are ways of partially coping with this but the basic lesson remains: life is not impossible, but it is exponentially difficult, and sometimes worse”. However, this goes beyond the scope of this course. $\square$

Example 45. With the above procedure, the branch of Example 43 would have just the root, left labelled by the formula, and the leaf $P(a)$. This branch defines the model with domain equal to $\{a\}$, and $P^J = \{a\}$.

2.3.4 Set Satisfiability and Entailment

Since a finite theory is unsatisfiable iff the conjunction of all the formulae of the theory is so, and $T \models \phi$ iff $T \cup \{\neg\phi\}$ is unsatisfiable (see Theorems 21 and 22), we can use tableaux for verifying the unsatisfiability of a finite set of formulae and the entailment of ϕ from a finite theory. The extension of the tableau procedure is defined as in the propositional case, with the proviso that the first order tableau procedure is not a decision procedure, see Note 13.

In the remainder, we reconsider the examples of Subsection 2.1.4. We leave out rules as right labels and square brackets, and ask students to add them whenever they are required.

Example 46. Consider the first-order language $\sigma := \langle R \rangle$ where R is a binary relation symbol. Let $T = \{\forall x R(x, x)\}$ be a σ -theory.

- Prove that $T \models \exists x R(x, x)$ using the tableau calculus.
- Prove that $T \cup \{\forall x \neg R(x, x)\}$ is unsatisfiable using the tableau calculus.



We prove that $T \models \exists x R(x, x)$ by building a tableau with only closed branches as follows.

$$\begin{array}{c}
 \forall x R(x, x) \\
 \neg \exists x R(x, x) \\
 | \\
 R(a, a) \\
 | \\
 \neg R(a, a) \\
 \text{closed}
 \end{array}$$

We prove that $T \cup \{\forall x \neg R(x, y)\}$ is unsatisfiable by building a tableau with only closed branches as follows.

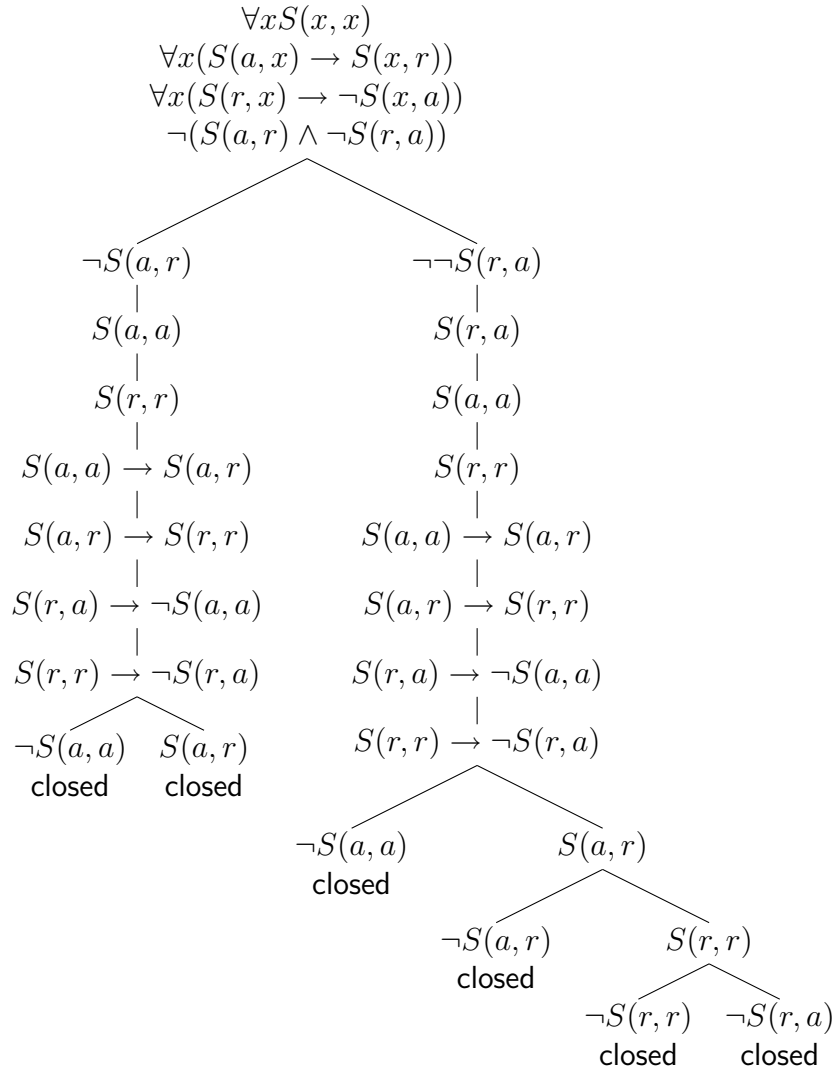
$$\begin{array}{c}
 \forall x R(x, x) \\
 \forall x \neg R(x, x) \\
 | \\
 R(a, a) \\
 | \\
 \neg R(a, a) \\
 \text{closed}
 \end{array}$$

Example 47 (Barber Story). Reconsider the revisited barber story. Our theory, T , is the following set of closed formulae

$$\left\{ \begin{array}{l} \forall x \quad (S(a, x) \rightarrow S(x, r)), \\ \forall x \quad (S(r, x) \rightarrow \neg S(x, a)), \\ \forall x \quad S(x, x) \end{array} \right\} \quad \begin{array}{l} (1) \\ (2) \\ (3) \end{array}$$

where: (1) stands for “if Angelo shaves an inhabitant then this shaves Roberto”; (2) stands for “if Roberto shaves an inhabitant then this does not shave Angelo”; (3) stands for “each inhabitant shaves himself”.

You are asked whether T entails the formalisation of “Angelo shaves Roberto and Roberto does not shave Angelo”, which is $S(a, r) \wedge \neg S(r, a)$. To determine this, we can use the tableau calculus on $T \cup \{\neg(S(a, r) \wedge \neg S(r, a))\}$ as below.



The tableau is closed hence $T \models S(a, r) \wedge \neg S(r, a)$.

Example 48 (Roncisvalle). Roncisvalle is a land of paladins. Astolfo and Rinaldo are two paladins of Roncisvalle.

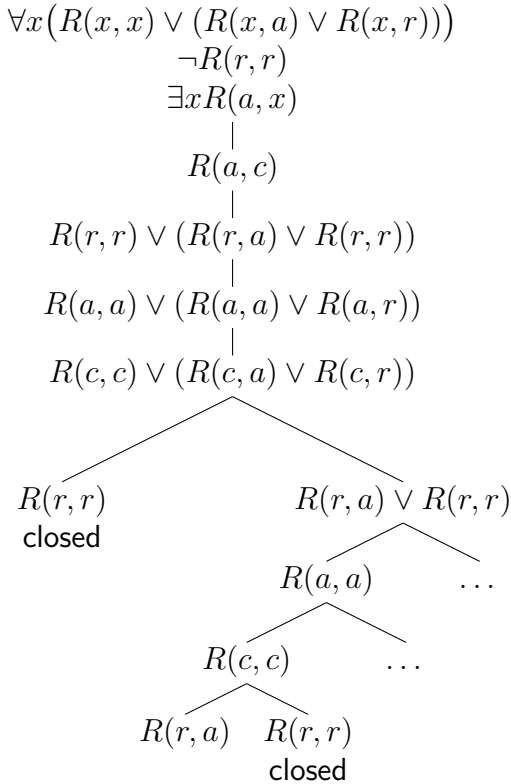
- (1). Each paladin decorates himself or decorates Astolfo or Rinaldo.
- (2). Rinaldo does not decorate himself.
- (3). Astolfo decorates a certain paladin.

(i) Using a suitable first order signature and the knowledge in (0), formalise (1), (2), (3). Let T be the resulting theory. (ii) Establish whether T is satisfiable or not.

The signature has two constant symbols, one for Rinaldo and the other for Astolfo; let r be the constant symbol for Rinaldo, and a the one for Astolfo. The signature will also need a binary relation symbol, say D , that stands for “to decorate”. In the signature, the sentences (1), (2), (3) are formalised as the following theory T :

$$\begin{array}{ll} (1) & \left\{ \forall x \ (R(x, x) \vee (R(x, a) \vee R(x, r))), \right. \\ (2) & \left. \neg R(r, r), \right. \\ (3) & \left. \exists x \ R(a, x) \right\}. \end{array}$$

We build a tableau as in the optimised procedure above and define a model for T with this procedure.



The middle branch is open and, following the optimised procedure, does not need to be expanded anymore. It defines the interpretation $\mathcal{I}$ with domain $D = \{a, c, r\}$, and $R = \{(a, c), (a, a), (c, c), (r, a)\}$. Check that this is a model of T —check it.

2.4 Summary

We saw that first order logic has: a language with its own grammar; a semantics; a proof theory.

2.4.1 Language

Given a signature, we defined the set of σ terms, atomic formulae, formulae, that is, $\text{FOL}(\sigma)$, and then zoomed in on the closed formulae of $\text{FOL}(\sigma)$. For every term and formula of σ we can generate a unique parsing tree; we can use parsing trees to recognise whether an expression of σ is a term/formula or not.

2.4.2 Semantics

- We defined interpretations $\mathcal{I}$ as tuples, with a domain and ways of interpreting constant, function and relation symbols in the domain. We defined $\mathcal{I} \models \phi$ for a generic *closed* formula $\phi \in \text{FOL}(\sigma)$; ways of reading this are “ $\mathcal{I}$ satisfies ϕ ” or “ $\mathcal{I}$ is a model of ϕ ”.
- We defined when a closed formula is satisfiable, unsatisfiable, valid, falsifiable.
- We extended the notion of satisfiability to sets of formula. We reduced the satisfiability of a finite set of formulae to the satisfiability of their conjunction.
- We extended the notion of validity to entailment, and reduced entailment of a formula from a finite set of formulae to satisfiability of a finite set of formulae.

2.4.3 Proof Theory

- We defined the notion of proof calculus in general, and distinguished between: natural deduction calculi, complete for entailment; computational calculi, refutation complete.
- We introduced the analytic tableau calculus for closed formulae.
 - We gave a tableau procedure:
 - * we proved that: if the input formula is unsatisfiable then the procedure terminates and returns a finite tableau with only closed branches; if the formula is satisfiable, the procedure returns a finite tableau with an open branch, or *at the limit* a tableau with an infinite open branch, that defines a model for the satisfiable formula;
 - * in other words, this gives us a semidecision procedure for verifying the validity of ϕ by building a tableau for $(\neg\phi)$; termination is not guaranteed in case the formula is not valid (Church theorem).

- We sketched an optimised version of the procedure.
- We extended this procedure to the verification of the satisfiability of a finite set of formulae, and hence to the entailment of a formula from finitely many formulae, giving examples of both—termination is not guaranteed, however, for the reasons above.

Bibliography

- [Barwise, J. and Etchemendy, J., 1991] Barwise, J. and Etchemendy, J. (1991). *The Language of First-Order Logic (2nd edn.)*. CSLI.
- [Bradley and Manna, 2007] Bradley, A. and Manna, Z. (2007). *The Calculus of Computation*. Springer.
- [Chiswell and Hodges, 2007] Chiswell, I. and Hodges, W. (2007). *Mathematical Logic*. Oxford University Press.
- [Fitting, 1997] Fitting, M. (1997). *First-Order Logic and Automated Theorem Proving*. Springer.
- [Huth and Ryan, 2010] Huth, M. and Ryan, M. (2010). *Logic in Computer Science*. Cambridge University Press.
- [Mendelson, 1997] Mendelson, E. (1997). *Introduction to Mathematical Logic*. Chapman and Hall.
- [Peltier, 2003] Peltier, N. (2003). A More Efficient Tableaux Procedure for Simultaneous Search for Refutations and Finite Models. In Cialdea Mayer, M. and Pirri, F., editors, *TABLEAUX 2003*, number 2796 in LNAI, pages 181–196. Springer.
- [Smullyan, 1968] Smullyan, R. (1968). *First-order Logic*. Springer.