

Logic: A Primer

Lecture Notes for Linguistics

Version of October 24, 2014

Erich H. Rast

Universidade Nova de Lisboa



Contents

Contents	i
1 Sets, Relations, Functions	1
1.1 Sets	1
1.1.1 Defining sets	1
1.1.2 Operations on Sets	5
1.1.3 Basic Properties of Sets	9
1.1.4 Venn Diagrams	9
1.1.5 Exercises	9
1.2 Relations	13
1.2.1 Ordered Tuples	13
1.2.2 Characterization of Relations	14
1.2.3 Other Important Notions	15
1.2.4 Properties of Relations	16
1.2.5 Exercises	19
1.3 Functions	20
1.3.1 Characterization of a Function	20
1.3.2 Properties of Functions	23
1.3.3 Further Notions	24
1.3.4 Exercises	26
1.4 Literature	28
2 Propositional Logic	31
2.1 Syntax of Propositional Logic	31
2.1.1 Basic Expressions	31
2.1.2 Well-Formed Formulas	33
2.1.3 Exercises	35

2.2	Semantics of Propositional Logic	36
2.2.1	Models and Truth in a Model	37
2.2.2	Truth Tables	38
2.2.3	Important Notions	41
2.3	Proof Theory	45
2.3.1	Tableaux Rules for Propositional Logic	47
2.3.2	How to Use Tableaux	48
2.3.3	Alternative Notation	52
2.3.4	Selected Theorems	52
2.3.5	Exercises	55
2.4	Deductive Arguments	55
2.4.1	Valid Argument Schemes	55
2.4.2	Sound Arguments, Fallacies, Good Arguments	56
2.4.3	Exercises	60
2.5	Metatheorems	62
2.6	Concluding Remarks	62
2.7	Literature	64
3	First-Order Logic	65
3.1	Syntax of First-order Predicate Logic with Identity	65
3.1.1	Basic Expressions.	65
3.1.2	Well-formed Formula.	66
3.1.3	Exercises	70
3.2	Semantics of First-Order Logic with Identity	70
3.2.1	Variable Assignments and Variants	70
3.2.2	Models and Truth in a Model	71
3.2.3	Explanation of the Rules for Predication and Quantification	73
3.2.4	Exercises	74
3.3	Proof Theory	74
3.3.1	Tableaux Rules for First-Order Predicate Logic	74
3.3.2	Rules for Identity	77
3.3.3	Using the Tableaux Rules	78
3.3.4	Selected Theorems	79
3.3.5	Exercises	81
3.4	Defined Notions	82
3.4.1	Russellian Descriptions	82
3.4.2	Relativized Quantifiers	83
3.4.3	Many-sorted Logic	84
3.4.4	The Existence Predicate	84
3.5	Applications to Natural Languages	85
3.5.1	Truth-Conditions and Pre-Montegovian Semantics	85
3.5.2	Some Problems	88
3.5.3	Deductive Arguments	91
3.5.4	Exercises	92

3.6	Metatheorems	94
3.7	Literature	95
4	Higher-Order Logic	97
4.1	Syntax of Simple Type Theory	97
4.1.1	Types	97
4.1.2	Terms	98
4.2	Semantics of Higher-Order Logic	100
4.2.1	General Models and Truth in a Model	100
4.2.2	Interdefinability of Quantifiers and Identity	101
4.2.3	More Definitions	101
4.3	Typed λ -Calculus	102
4.3.1	Conversion Rules	102
4.3.2	λ -Abstraction at Work	103
4.3.3	Exercises	104
4.4	Applicative Categorical Grammar	105
4.4.1	Introduction	105
4.4.2	Type-driven Evaluation	107
4.5	Applications	111
4.5.1	Verbs, Proper Names	111
4.5.2	Generalized Quantifiers	111
4.5.3	Generalized Quantifiers and the Finite Verb Phrase	113
4.5.4	Quantifier Scope Ambiguities	114
4.5.5	Outlook and Limits	115
4.6	Metatheorems	116
	Solutions to Exercises	121
	Index	129

Preface

This text is a short introduction to logic that was used for accompanying an introductory course in Logic for Linguists held at the New University of Lisbon (UNL) in fall 2010.

The main idea of this course was to give students the formal background and skills in order to assess literature in logic, semantics, and related fields and perhaps even use logic on their own for the purpose of doing truth-conditional semantics. This course in logic does not replace a proper introduction to semantics and is not intended as such, although parts of Chapter 1 and 4 could be used to supplement an introductory course in semantics. In contrast to other introductions it has a certain focus on ‘writing things down correctly’, which is why not always the simplest notation is used—for example, Greek letters are used as metavariables to encourage students to learn the Greek alphabet. However, proofs of metatheorems have been omitted entirely.

This text is not recommend for self-study, because it is in (eternal?) draft status and still contains errors and typos. For self-study it is better to rely on material that has been reviewed more extensively. Please report typos and errors to erich@snafu.de. This manuscript was written hastily by a non-native speaker, and so I’d like to apologize for any odd uses of the English language.

Erich Rast,
15. December 2011

The Greek Alphabet

α	A	alpha
β	B	beta
γ	Γ	gamma
δ	Δ	delta
ϵ	E	epsilon
ζ	Z	zeta
η	H	eta
θ	Θ	theta
ι	I	iota
κ	K	kappa
λ	Λ	lambda
μ	M	mu
ν	N	nu
ξ	Ξ	xi
\omicron	O	omicron
π	Π	pi
ρ	P	rho
σ	Σ	sigma
τ	T	tau
υ	Y	upsilon
ϕ	Φ	phi
χ	X	chi
ψ	Ψ	psi
ω	Ω	omega

List of Symbols

\emptyset	empty set
$a \in B$	membership, a is a member of B
$A \cup B$	union of A and B
$A \cap B$	intersection of A and B
$A \subset B$	proper subset, A is a proper subset of B
$A \subseteq B$	(improper) subset, A is a subset of B
$A \setminus B$	set difference, A without B
\overline{A}	complement of A
$\mathcal{P}(A)$	powerset of A
A^n	n -ary Cartesian product of A
$A_1 \times \cdots \times A_n$	Cartesian Product of A_1, \dots, A_n
B^A	the functions from A to B
$f : A \rightarrow B$	function f from A to B
$a \mapsto b$	a is mapped by a function to b
f^{-1}	inverse function of f
R^{-1}	inverse relation of R
$=$	identity
\sim	equivalence relation
\geq	greater than or equal
$>$	greater than
\neg	truth-functional negation
\vee	(inclusive) disjunction
$\dot{\vee}$	exclusive disjunction
\wedge	conjunction
\rightarrow	conditional
\leftrightarrow	biconditional

\uparrow	Sheffer stroke
\downarrow	Peirce stroke
$Cn(.)$	deductive closure
$M \models \phi$	ϕ is true in M
$\models \phi$	ϕ is a valid
$\vdash \phi$	ϕ is provable
$\phi_1, \dots, \phi_n \vdash \psi$	ψ is provable from ϕ_1, \dots, ϕ_n
$\llbracket \phi \rrbracket^{M,g}$	evaluation of ϕ in M under assignment g
\forall	universal quantifier
\exists	existential quantifier
ι	iota operator
ι	iota quantifier
$\exists!$	there is exactly one
$g[x/a]$	modified assignment, s.t. $g(x) = a$
$\phi[x/t]$	replace free x by t in ϕ
$g \approx_x h$	x -variant h of assignment g
λ	λ -abstraction operator
$\phi \Rightarrow \psi$	rewrite ϕ as ψ
$\phi \Leftrightarrow \psi$	$\phi \Rightarrow \psi$ and $\psi \Rightarrow \phi$

Sets, Relations, Functions

The topic of this chapter are sets, relations, and functions. It is very likely that you already have a fairly good idea of what these are from your inevitable exposure to school mathematics, in case of which you may use this chapter and the easy exercises in it to refresh your knowledge a bit. Set theory is a necessary prerequisite to formulating logical languages and models for them in a modern way and assess literature in logic and formal semantics. Even if you think you already know what functions are you ought not skip this chapter in its entirety for two reasons. First, the notation used here might differ from the one you're used to and the way sets, relations, and functions are used throughout here might differ from what you already know. Second, we will take a look at a number of linguistic examples that you might find interesting.

1.1 Sets

Sets are abstract collections of objects.

1.1.1 Defining sets

When characterizing a set you need to be precise and avoid ambiguities of natural language. In the literature, sets are often defined using notation from mathematics and logics, but the level of formality depends on the style of the author and many great mathematicians and logicians have the ability to express themselves precisely with only a minimum amount of 'notational clutter'. There is always a tension between notational correctness and readability, and the goal is to find the right equilibrium between them. That being said, let us take a look at the ways to define sets. English paraphrases are given in quotes; they also represent how you would read those definitions aloud.

Enumeration. All the members of the set are listed within curly braces { and }, where list items are separated by comma.

- $\{1, 2, 3, 4\}$
“the set containing the numbers 1, 2, 3, 4”
- $A = \{Peter, John, Mary\}$
“the set A contains Peter, John, and Mary (and nothing else)”
- $C := \{1, 2, 3, \dots, 20\}$
“let C be a set containing the integer numbers from 1 to 20”

☞ **Note 1 (Identity versus Definition)** The sign ‘:=’ indicates a definition, which ought to be understood as a mere syntactic abbreviation. In contrast to this ‘=’ denotes identity.

Abstraction, Set-builder Notation. A variable – usually x, y, z – is used in combination with a condition that this variable must satisfy.

- $A := \{x \mid x \text{ is a natural number}\}$
“A is the set of all natural numbers”
- $B := \{x \mid x \text{ is a male person}\}$
“B is the set of all male persons”
- $C := \{x \mid x \in \mathbb{N} \text{ and } x \geq 20\}$
“C is the set of all natural numbers that are greater than or equal to 20”
- $D := \{s \mid \text{someone jumps in } s \text{ at the time of } s\}$
“D is the set of jumping situations”

Hereby the notation $x \in \mathbb{N}$ means that x is a member of the set \mathbb{N} , which by convention denotes the set of natural numbers. Sometimes the domain of the variable is specified in the first part of the definition using the membership relation \in (explained in more detail below).

- $\{x \in \mathbb{R} \mid x > 0\}$
“the set of real numbers greater than 0”
- $\{x \in \mathbb{N} \mid \text{there is an } k \in \mathbb{N} \text{ such that } x = 2k\}$
“the set of even natural numbers”

Recursive Definitions. Sets can be built from more basic sets by a recursive definition:

- $t \in A$
“ t is an element of A ”
- If $\alpha \in A$ and $\beta \in A$, then $(\alpha\beta) \in A$
- Nothing else is in A

Giving a Precise Description. Sets are also often just described precisely without using any special notation:

- Let A be the set of all prime numbers.
- B is the set of positive natural numbers greater than 2

Empty Set. The empty set \emptyset is the set that does not contain any element. There is only one empty set. Its existence is either postulated explicitly or can be inferred in common axiomatic formulations of set theory.

Q History 1 The symbol ‘ \emptyset ’ was introduced by the Bourbaki Group, an influential group of mathematicians publishing under the pseudonym ‘Nicolas Bourbaki’ starting from 1935. The symbol is based on the letter \emptyset used in Danish and Norwegian.

It should be clear that just defining a set by abstraction doesn’t give you any guarantee that the set actually has members. The members of sets stand for concrete entities, but the sets themselves are abstract entities and not aggregates of actual objects. You can see this from the fact that the empty set is itself a set. Someone may for example clearly and unambiguously describe the set of coins he has in his left pocket and yet it may happen that he has no coins in his pocket – and a collection of 0 coins is not an aggregate of any actual coins!

When a set A has members, i.e. $A \neq \emptyset$, you have to think of these members as the actual objects and not symbols standing for them unless we’re talking about a set of symbols in the first place. For example, consider a set $\{Pedro, Afonso, Maria\}$; this set consists of Pedro, Afonso, and Maria and not of the proper names ‘Pedro’, ‘Afonso’, and ‘Maria’. In other words, the names are used and not mentioned. Of course, in linguistics it is also common to talk about sets of symbols and in that case care has to be taken to make clear that the objects in question are symbols. For example, $\{\text{‘Pedro’}, \text{‘Afonso’}, \text{‘Maria’}\}$ is the set of proper names ‘Pedro’, ‘Afonso’, and ‘Maria’.

The actual members of a set are sometimes called its *extension*. The empty set has an empty extension, because it has no members. When a set is described or characterized using set-builder notation, we can think of the description of the set as characterizing the set's *intension* apart from describing its extension. Take for example the two sets $A := \{x \mid x \text{ is a being that has a liver}\}$ and $B := \{x \mid x \text{ is a being that has a heart}\}$. For all we know A and B have the same extension: There is no being with a liver that has no heart and there is no being with a heart that has no liver. Nevertheless, A and B have different intensions. In order to show that A and B have the same extension you have to look at all beings with livers and hearts; you cannot infer this from the definitions of the sets alone.

You may also look at the distinction in the following way: Suppose John is a bit crazy and makes a list of all beings with a heart by naming not just their species but each of them individually. Then by just looking at this huge list and investigating the world you could not figure out whether he wanted to give you the set of all beings with a liver, the set of all beings with a heart, the set of all beings with a liver or a heart, or the set of all beings with a liver and a heart. An intension contains more information than the extension. However, there are many different ways to formally capture the notion of an intension, each of them has its quirks, and there is a long history of philosophical controversies about the notion of an intension. For the time being, we will not attempt to give a precise account of what an intension is and instead characterize it negatively: whatever is not clearly extensional is intensional. This rule of thumb is tied to the *identity conditions* associated with a certain kind of entity. Even though they can be defined by abstraction in terms of their intension sets are extensional in the sense that only their members are taken into account when comparing them.

Identity Between Sets. Two sets A and B are identical, i.e. $A = B$, if all members of A are also members of B and all members of B are also members of A .

☞ **History 2 (Extension versus Intension)** Many attempts of implementing intensions in a logical language were inspired by Gottlob Frege's work, in particular his well-known article 'Über Sinn und Bedeutung' (1892). Frege distinguishes between the sense of an expression, which corresponds to what is nowadays called an intension, and its reference, which corresponds to the extension.

The above identity condition is a deliberate choice. There are logical systems in which entities that are in many ways similar to sets do not have an extensionality

principle like the above one. Notice that from the definition of identity between sets it follows that $\{a, a, b, a\} = \{a, b\} = \{b, a\}$. Duplicate entries don't count and are usually omitted and the order to the elements in the specification of a set doesn't matter. Notice further that $\emptyset = \emptyset$ is a special case. All members of \emptyset are also members of \emptyset simply because \emptyset has no members. This way of understanding 'all' is based on mathematical conventions.

☞ **Note 2 (Presuppositional Readings of Quantifiers)** In natural languages quantifiers are often ambiguous between a strict 'logical' reading and a *presuppositional reading*. Consider the following example:

(1.1) *Situation: Pedro is talking to Maria about the homework assignments. Next to him is a small wooden box that is closed. The box is empty. He is pointing to the box and utters:*

If you make this exercise for me, I'll give you all the money in this box.

According to the non-presuppositional reading there is nothing wrong with Pedro's suggestion. In this reading 'all the money in this box' can denote the empty set. However, when somebody uses the English quantifier 'all' it is often silently presupposed that there is at least one object satisfying the quantifier restriction, i.e. satisfying the property *being money in this box* in the above example.¹The presupposition that there is an object satisfying the quantifier restriction is not commonly made in mathematical parlance.

1.1.2 Operations on Sets

Membership. $a \in D$ expresses the fact that a is a member of the set D . If $a \in D$ we also say that D contains a , that a is an element of D , or simply that a is in D . The empty set is a member of any set.

Intersection. $A \cap B$ denotes the set containing all elements that are in A and in B , i.e. $A \cap B := \{x \mid x \in A \text{ and } x \in B\}$. $A \cap B$ is called *the intersection of A and B*.

Union. $A \cup B$ denotes the set containing all elements that are in A or in B or in both, i.e. $A \cup B := \{x \mid x \in A \text{ or } x \in B\}$. $A \cup B$ is called *the union of A and B*.

Subset. $A \subseteq B$ holds if and only if all elements of A are elements of B . If $A \subseteq B$ is the case we say that A is a subset of B .

¹ This presupposition can also be explained by using Gricean conversational maxims. Do you know how?

Proper Subset. A is a proper subset of B , written $A \subset B$, if and only if A is a subset of B and B is not a subset of A , i.e. $A \subseteq B$ and *not* $B \subseteq A$.²

Set Difference and Complement. The *difference* of two sets A, B is written $A \setminus B$, spoken *A without B* or *A minus B*, and contains all members of A that are not in B . When $B \subseteq A$ is the case $A \setminus B$ is also called *the complement of B in A*. $A \setminus B := \{x \in A \mid x \notin B\}$. Alternative notation: $A - B$. Sometimes a base domain D such that $A \subseteq D$ is known or presumed and then \overline{A} is used to denote the complement of A in D , i.e. $\overline{A} := \{x \in D \mid x \notin A\}$ where D must be clear from previous definitions or easily inferable from the context.

Cardinality. The cardinality of a set is a measure of its size. It is usually written $|A|$. For example, for $A = \{a, b, c\}$, $|A| = 3$. Obviously, $|\emptyset| = 0$.

✧ **Remark 1 (Cardinality and Infinity)** The cardinality of the set of natural numbers \mathbb{N} is called \aleph_0 (pronounced: ‘aleph null’; \aleph is a letter of the Hebrew alphabet), standing for infinitely many elements that can be counted. Such a set is said to be *countable* or *denumerable*. The cardinality of the set of real numbers \mathbb{R} is 2^{\aleph_0} . This cardinal number, called the cardinality of the continuum, is distinct from \aleph_0 , because it can be proved that there is no one-on-one mapping from the set of real numbers to the set of natural numbers. This means that even though both sets contain infinitely many numbers there are more real numbers than natural numbers.

📖 **History 3 (Georg Cantor (1845-1918))** Among the numerous important results by Georg Cantor was a famous proof, using a technique called diagonalization, that there are more real numbers than natural numbers. He also was the first to advance the *continuum hypothesis*, which states that there is no set with a cardinality that lies between the cardinality of the set of natural numbers and the cardinality of the set of real numbers. In modern notation this would be expressed as the hypothesis that $\aleph_1 = 2^{\aleph_0}$. This hypothesis was conjectured in 1897; it is still unproved but widely accepted to be true.

² It is a general convention to strike through symbols to indicate their negation, so the second part of the above condition could have been written $B \not\subseteq A$.

*** Example 1** Consider a situation with the following domain $D := \{\text{Pedro, Afonso, Maria, Ana, Elisabeta, Erich, Rui}\}$. Suppose, for example, D represents the persons in a classroom at a given time. Using a bit non-standard notation, let $students := \{\text{Pedro, Afonso, Maria, Ana, Elisabeta}\}$, $yawn := \{\text{Erich, Ana}\}$, $laugh := \{\text{Maria, Afonso}\}$, and $work := \{\text{Pedro, Afonso, Maria, Ana, Elisabeta, Rui}\}$

(1.2) Erich and Rui are *not* students:

$$\overline{student} = \{\text{Erich, Rui}\}$$

(1.3) No one is laughing *and* yawning:

$$yawn \cap laugh = \emptyset$$

(1.4) All students work:

$$students \subseteq work$$

(1.5) The set of students that laugh *or* yawn:

$$laugh \cup yawn = \{\text{Maria, Afonso, Erich, Ana}\}$$

(1.6) The set of students that do *not* laugh:

$$students \setminus laugh = \{\text{Pedro, Ana, Elisabeta}\}$$

(1.7) All *laughing people* (in the classroom, at the given time) are students:

$$laugh \subseteq students$$

(1.8) *Some* students yawn:

$$(students \cap yawn) \neq \emptyset$$

Note 3 (Generalized Quantifiers) Natural language expressions like ‘todos estudiantes’, ‘drei Könige’ (three kings), or ‘some linguists’ are from a semantic perspective considered *generalized quantifiers*. Correspondingly, formulations of the meanings of expressions like ‘todos’, ‘drei’, and ‘some’ used in these generalized quantifiers are sometimes called *generalized (quantifying) determiners*. In a phrase like ‘Some students yawn’ the expression ‘some’ is a generalized quantifying determiner, the plural NP ‘students’ is the quantifier restriction, and the finite verb phrase ‘yawn’ is the body of the quantifier. Ignoring for the time being the intricacies of the syntax-semantics interface, it is easy to give truth-conditions for many generalized determiners using set theory. Here are some examples, where A stands for the meaning of the quantifier restriction and B for its meaning of the quantifier body both taken as sets of objects:

- some A B: $(A \cap B) \neq \emptyset$
Example: Some philosophers are linguists.

- all A B: $A \subseteq B$
Example: All linguists hate philosophy.
- no A B: $(A \cap B) = \emptyset$
Example: No student likes logic.
- most A B: $|(A \cap B)| > |(A \setminus B)|$
Example: Most linguists like syntax.
- three₌ A B: $|(A \cap B)| = 3$
Example: Three dogs bark.

A side note about ‘three’. I have marked this definition with a = in order to indicate that according to this definition the quantifying determiner is read ‘exactly three’. This reading seems to be prevalent.³ But in certain circumstances it may also be possible to understand numerals like ‘three’ as in ‘ x or more’, i.e. based on the definition $|(A \cap B)| \geq x$. For example, when it is known by everyone in a conversation that a family with two or more children benefits from reduced taxes an utterance of ‘Maria has two children, so she’ll get tax benefits’ seems to be perfectly true if Maria has four children. Or what do you think? What about Portuguese numerals? Do they have the same default interpretation?

Powerset. The powerset $\mathcal{P}(A)$ of a set A is the set of all subsets of A , i.e. $\mathcal{P}(A) = \{X \mid X \subseteq A\}$. Alternative notations for the power set are $P(A)$, $\wp(A)$, or 2^A . Notice that $\emptyset \in \mathcal{P}(A)$ and $A \in \mathcal{P}(A)$ for any set A .

✧ **Remark 2 (The Power of the Powerset)** As the name suggests the powerset operation is very powerful. Generally, the powerset of a set with n elements has 2^n elements. Recall that the set of natural numbers \mathbb{N} is countably infinite, i.e. $|\mathbb{N}| = \aleph_0$. The powerset $\mathcal{P}(\mathbb{N})$ thus has cardinality 2^{\aleph_0} , i.e. it has the cardinality of the set of real numbers. This implies that the members of $\mathcal{P}(\mathbb{N})$ are not countable.

³ Once more Gricean maxims can give an explanation as to why this reading is prevalent. Bear in mind, though, that the question whether this reading is prevalent or not is empirical and premature judgments ought to be avoided.

1.1.3 Basic Properties of Sets

Here are a few propositions that hold for any sets A, B :

- | | |
|---|---|
| 1. $\emptyset \subseteq A$ | 7. $(A \cup B) = (B \cup A)$ |
| 2. $(A \cup \emptyset) = A$ | 8. $(A \cap B) = (B \cap A)$ |
| 3. $(\emptyset \cup \emptyset) = \emptyset$ | 9. $((A \cup B) \cup C) = (A \cup (B \cup C))$ |
| 4. $(A \cap \emptyset) = \emptyset$ | 10. $A \subseteq (A \cup B)$ |
| 5. $(A \cap A) = A$ | 11. $(A \cap B) \subseteq A$ |
| 6. $(A \cup A) = A$ | 12. $\overline{(A \cap B)} = \overline{(A \cup B)}$ |

1.1.4 Venn Diagrams

Venn diagrams are helpful tools for visualizing relations between sets. Examples of Venn diagrams are given in Figure 1.1 to 1.5.

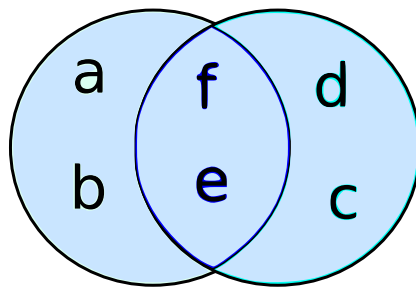


Figure 1.1: Union of sets $A = \{a, b, f, e\}$ and $B = \{d, c, f, e\}$.

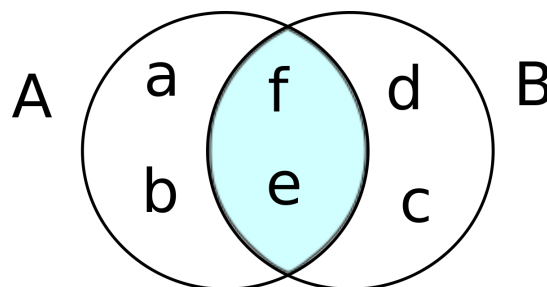


Figure 1.2: Intersection of sets $A = \{a, b, f, e\}$ and $B = \{d, c, f, e\}$.

1.1.5 Exercises

 **Exercise 1** Define the following sets by enumeration:

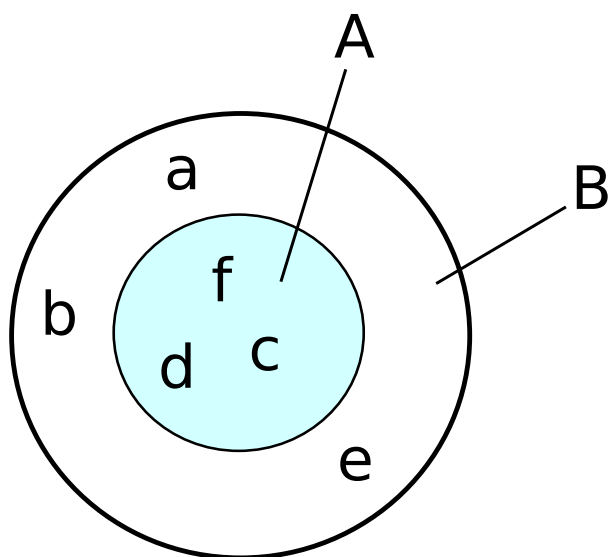




Figure 1.3: Set $A = \{f, d, c\}$ is a proper subset of $B = \{a, b, c, d, e, f\}$.

- a. all possible outcomes of a single throw of a standard dice
- b. all faces that of a single throw of two dices might show, where it is not distinguished between the dice (for example, the case when the first dice shows 1 and the second 3 is treated as being equal to the case when the second dice shows 1 and the first dice shows 3)

 **Exercise 2** Define the following sets by specifying their intension, using set abstraction:

- a. the set of odd natural numbers greater than 3
- b. the set of all subsets of a set S
- c. the set of all blue sports cars in Lisbon today
- d. the set of all subsets of a set A whose intersection is non-empty

 **Exercise 3** Let $A := \{1, 3, 4, 5, 2\}$, $B := \{a, 3, 4, 5, b, c\}$, $C := \{1, 9\}$. Determine the following sets by enumerating their members:

- | | |
|------------------------|---|
| a. $(A \cup B) \cap C$ | c. $(A \setminus C) \cap B$ |
| b. $(A \cap B) \cup C$ | d. $(C \setminus A) \cup ((B \cap A) \cup \emptyset)$ |

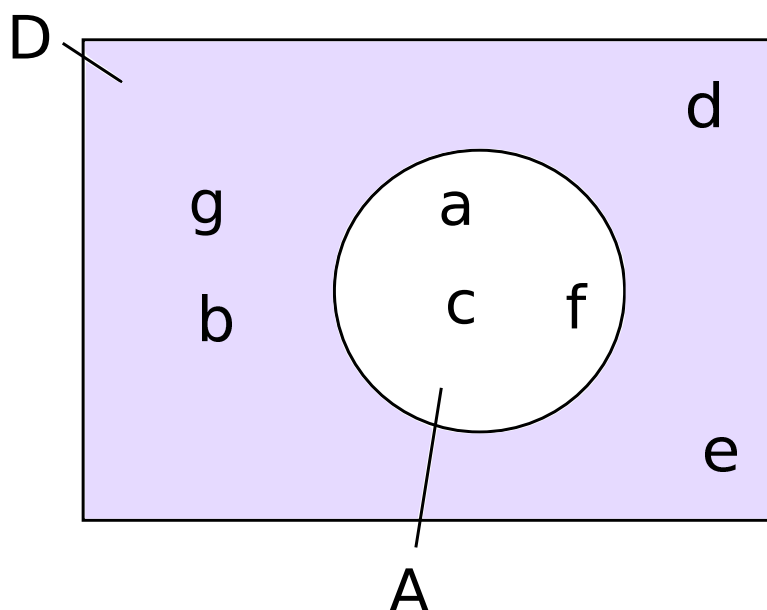


Figure 1.4: Let $A = \{a, c, f\}$ and the domain $D = \{g, b, d, e, a, c, f\}$. Then $\bar{A} = \{g, b, d, e\}$.

 **Exercise 4**

- Does the phrase ‘the first three members of the set $\{a, b, c, d, e, f, g\}$ ’ make sense? Explain!
- Does it make sense to speak of the union of a given set of apples and a given set of bananas? Explain!
- Express the phrase ‘All employees that are not members of the union get a higher salary’ in the language of set theory.
- Express the phrases ‘Há estudantes que trabalham’ and ‘Há estudantes que não trabalham’ in the language of set theory.
- Express the phrase ‘Todos estudantes trabalham ou não trabalham’ in the language of set theory.
- Is the phrase of the previous exercise (e) always true or can it also be false?
- Express the fact that the intersection of three sets A, B, C is non-empty in the language of set theory.
- Suppose a set A is empty and a set B is non-empty. Does $A \subseteq B$ hold?

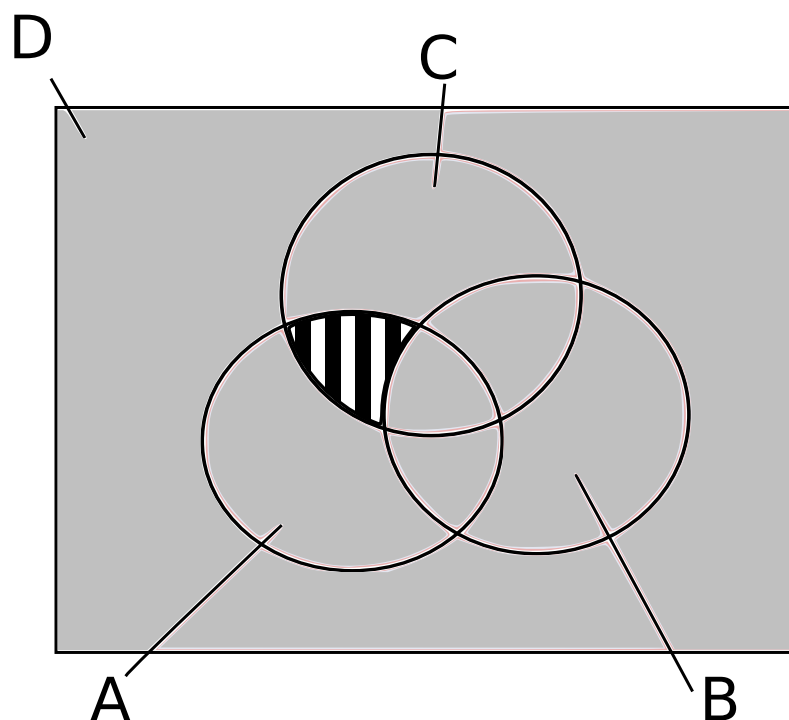



Figure 1.5: The hatched area depicts $(A \setminus B) \cap C$. The grey area depicts the complement $\overline{(A \setminus B) \cap C}$ in D .

✎ **Exercise 5** Let $A := \{a, b, c\}$ and $B := \{d, e, a\}$ and let the total domain D under consideration be $A \cup B \cup \{f\}$. Draw Venn diagrams to illustrate whether or not the relations between the following sets hold:


- | | |
|--------------------------|--|
| a. $A \cup B$ | e. $D \setminus (A \cup B)$ |
| b. $\overline{A} \cup B$ | f. $A \subset \overline{B}$ |
| c. $\overline{A \cap B}$ | g. $A \subseteq B$ and $B \subseteq A$ |
| d. $D \setminus \{f\}$ | |

✎ **Exercise 6** Use Venn diagrams to show whether or not the following propositions hold:

- $A \subseteq (A \cap B)$
- If $(A \cup B) \subseteq B$ then $A \subseteq B$.
- $\overline{(\overline{A \cup B})} = (A \cap B)$

 **Exercise 7** Specify the answer to the following questions by enumerating all members of the answer set.

- a. $A := \{1, 2\}$. What is $\mathcal{P}(A)$?
- b. $A := \emptyset$. What is 2^A ?
- c. $A := \{a, b, c\}$ and $B := \{c, a\}$. What is $\mathcal{P}(A \cap B)$?

 **Exercise 8** Formulate truth conditions like in Note 3 for the following generalized determiners:

- | | |
|----------------------|---------------------------|
| a. at least five A B | c. no more than three A B |
| b. exactly one A B | d. not one A B |

1.2 Relations

We will now take a closer look at properties and relations. A property can formally be represented by its extension: the set of objects that have the property. For example, the property of being a student may be represented by the set of all students and the property of being a record produced between 1970 and 1980 and liked by Erich can be represented as one huge, yet still finite, countable, and enumerable set of records. But can we also represent relations between objects extensionally? The answer is, of course, Yes. The extension of an n -ary relation can be given by a set of ordered n -tuples. An ordered 2-tuple is called an *ordered pair*.

1.2.1 Ordered Tuples

Ordered Pair. We write $\langle a, b \rangle$ for the ordered pair consisting of a in the first place and b in the second place. Notice that, as the name implies, the order now matters. This means that if $a \neq b$, then $\langle a, b \rangle \neq \langle b, a \rangle$. Bear in mind, however, that the case that $a = b$ might sometimes have to be taken into account. The notation (a, b) is also sometimes used for an ordered pair.

Ordered n -Tuple. We write $\langle a, b, c \rangle$ for the ordered triple consisting of a , b , c (in that order), write $\langle a, b, c, d \rangle$ for the ordered quadruple consisting of a , b , c , and d (in that order), write $\langle a, b, c, d, e \rangle$ for the ordered quintuple consisting of a , b , c , d , and e (in that order), and generally $\langle a_1, a_2, \dots, a_n \rangle$ for the ordered n -tuple consisting of a_1, a_2, \dots, a_n (in that order).⁴

⁴ Ordered pairs can be defined using set theory by representing $\langle a, b \rangle$ by $\{a, \{b\}\}$, $\langle a, b, c \rangle$ by $\{a, \{c, \{c\}\}\}$, and so on. These tricks do not matter for our purpose, but they are sometimes used for the definition of list data structures in programming languages.

1.2.2 Characterization of Relations

Relation and Arity. A relation between two sorts of entities is called a *binary relation*. A relation between three sorts of entities is called a *ternary relation*. The number of arguments of a relation is called its *arity*. So we can generally speak of n -ary relations ($n \geq 1$). Notice that unary predicates have arity 1 and are sometimes considered a special case of a relation.

Sometimes authors suggest that an n -ary predicate ($n \geq 2$) expresses a relation similar to saying that a unary predicate expresses a property. We avoid this way of talking, because it carries some perhaps undesirable ontological baggage. Instead, we use the term ‘relation’ sometimes for the symbol or expression like ‘ R ’ or ‘to know’ and sometimes for its meaning: Something that holds between two or more objects. The term ‘predicate’ is here mostly used in the context of talking about unary predicates (i.e. of arity 1). If only syntactic entities like ‘ P ’ or ‘ R ’ are meant we can make this explicit by calling them predicate symbols and relation symbols respectively. In chapter 3 the connection between predicate and relation symbols and their extensional meaning will be made precise by specifying concrete interpretation rules that map *prima facie* meaningless and arbitrary symbols to their extension, thereby specifying their meaning in a mathematically precise way.

Extensional Representation of a Relation. The extension of an n -ary relation can be represented by a set of n -tuples of entities.

✱ **Example 2** Here are some examples of relations:

(1.9) \geq is a binary relation between two numbers

(1.10) As long as tense and aspect is ignored, giving something to someone is a ternary relation between an agent, the recipient, and the object that is given.

$\{\langle x, y, z \rangle \mid x \text{ gives } y \text{ to } z \text{ in Sala 4.02 on October 24, 2014}\}$

(1.11) $x \leq (y + z)$ is a ternary relation between numbers⁵

$\{\langle x, y, z \rangle \mid x \leq (y + z)\}$

(1.12) As long as tense and aspect is ignored, buying something from someone is a relation of arity 4: A buyer buys something at a certain price from a seller.

$\{\langle x_1, x_2, y, z \rangle \mid x_1 \text{ buys } x_2 \text{ from } y \text{ at price } z\}$

⁵ I’m using variables x, y, z for indicating arguments in this example. More about this will be said in the following sections.

1.2.3 Other Important Notions

Cartesian Product. The Cartesian product $A \times B$ between two sets A and B is the set of all ordered pairs $\langle x, y \rangle$ such that $x \in A$ and $y \in B$, i.e. $A \times B = \{\langle x, y \rangle \mid x \in A \text{ and } y \in B\}$. By convention \times is also commonly used for specifying the set of n -tuples of several sets A_1, A_2, \dots, A_n ($n > 2$).⁶

✱ **Example 3 (Cartesian Product)** (1.13) Let $A := \{a, b, c\}$, $B := \{e, f\}$.

Then $A \times B = \{\langle a, e \rangle, \langle b, e \rangle, \langle c, e \rangle, \langle a, f \rangle, \langle b, f \rangle, \langle c, f \rangle\}$.

(1.14) Let $A := \{x \mid x \text{ is a positive integer}\}$ and $B := \{x \mid x \text{ is a positive even integer}\}$. Then $A \times B = \{\langle x, y \rangle \mid x \text{ and } y \text{ are positive integers and } y \text{ is even}\}$.

(1.15) $A := \{Mary, John\}$ and $B := \{John, Peter\}$. Then

$A \times B := \{\langle Mary, John \rangle, \langle Mary, Peter \rangle, \langle John, John \rangle, \langle John, Peter \rangle\}$.

Extensions of Predicates and Relations. Suppose there is a fixed domain D of objects we're talking about. Then the extension of a predicate is a member of $\mathcal{P}(D)$, the extension of a binary relation is an element of $\mathcal{P}(D \times D)$, the extension of a ternary relation is an element of $\mathcal{P}(D \times D \times D)$, and so forth.

Inverse Relation. If R is an n -ary relation, then its inverse relation R^{-1} is $\{\langle x_1, \dots, x_{n-1}, x_n \rangle \mid R(x_n, x_{n-1}, \dots, x_1)\}$. Notice that we used the notation $R(x, y)$ for indicating that x stands in relation R to y . Later we will make this usage more precise by defining a formal language with relation symbols like R and variables like x, y, z , which are interpreted correspondingly as relations and individuals on a base domain. For the time being, let us adopt this notation as a shortcut. Alternatively, we could have taken R to directly stand for its extension and formulated the condition as $R^{-1} := \{\langle x_1, \dots, x_n \rangle \mid \langle x_n, x_{n-1}, \dots, x_1 \rangle \in R\}$.

✱ **Example 4** Consider the relation 'x knows y'. Suppose we have a group D of people, say $D = \{Maria, Pedro, Anna\}$. We know that the extension of our relation is an element in $\mathcal{P}(D \times D)$, i.e. a subset of all ordered pairs on D . Perhaps nobody in the group knows anyone else in the group. Then the extension is the empty set. (Recall that $\emptyset \in \mathcal{P}(A)$ for any set A , no matter what it contains.) Let's assume instead that the extension of 'knows' in our situation (with respect to D) is $\{\langle Maria, Anna \rangle, \langle Anna, Maria \rangle, \langle Maria, Maria \rangle, \langle Pedro, Anna \rangle, \langle Pedro, Pedro \rangle\}$. Anna does not know herself – let us set aside for the moment the interesting question whether it is adequate to represent this fact in the above way or whether perhaps knowing oneself has a different meaning than knowing someone and ought to be a relation on its own.⁷

⁶ Notice that for example $A \times (B \times C)$ can be used for $A \times B \times C$. Even though $\langle a, b, c \rangle$ and $\langle a, \langle b, c \rangle \rangle$ are not the same entities, they can be used for the same purpose and sometimes this is done silently. Likewise, a 1-tuple $\langle a \rangle$ is in many applications silently taken as being equal to a itself, even though this is formally not quite correct.

⁷ What's your opinion?

The inverse relation in this case can be paraphrased as ‘ x is known by y ’ and its extension is $\{\langle Anna, Maria \rangle, \langle Maria, Anna \rangle, \langle Maria, Maria \rangle, \langle Anna, Pedro \rangle, \langle Pedro, Pedro \rangle\}$. Since Maria knows Anna, Anna is known by Maria and since Pedro knows Anna, Anna is also known by Pedro. Maria and Pedro also know themselves and hence are known by themselves. But Anna does not know Pedro, even though she is known by him.

1.2.4 Properties of Relations

In the following list I will give the alternative symbolic notation in first-order predicate logic. This logical language will be introduced in the next part, so for the time being you may ignore this notation if you’re not yet familiar with it.

Common Properties of Relations.

1. A relation R is **reflexive** if and only if $R(x, x)$ for all x .
 $\forall x[R(x, x)]$
2. A relation R is **irreflexive** if and only if for all x it is not the case that $R(x, x)$. Notice that this is *not* the same as just not being reflexive!
 $\forall x[\neg R(x, x)]$
3. A relation R is **symmetric** if and only if for all x and y : If $R(x, y)$ then $R(y, x)$.
 $\forall x, y[R(x, y) \rightarrow R(y, x)]$
4. A relation R is **antisymmetric** if and only if for all x and y : If $R(x, y)$ and $R(y, x)$ then $x = y$. Notice that this is *not* the same as not being symmetric!
 $\forall x, y[(R(x, y) \wedge R(y, x)) \rightarrow x = y]$
5. A relation R is **asymmetric** if and only if for all x and y : If $R(x, y)$ then it is not the case that $R(y, x)$.
 $\forall x, y[R(x, y) \rightarrow \neg R(y, x)]$
6. A relation R is **transitive** if and only if for all x, y , and z : If $R(x, y)$ and $R(y, z)$ then $R(x, z)$.
 $\forall x, y, z[(R(x, y) \wedge R(y, z)) \rightarrow R(x, z)]$
7. A relation R is **Euclidean** if and only if for all x, y , and z : If $R(x, y)$ and $R(x, z)$ then $R(y, z)$.
 $\forall x, y, z[(R(x, y) \wedge R(x, z)) \rightarrow R(y, z)]$
8. A relation R is a **preorder** (quasi-order) if and only if R is reflexive and transitive.
9. A relation R is **total** if and only if for all x and y : $R(x, y)$ or $R(y, x)$.
 $\forall x, y[R(x, y) \vee R(y, x)]$

10. A relation R is a **partial order** if and only if R is reflexive, antisymmetric, and transitive.
11. A relation R is a **total order** if and only if R is total, reflexive, antisymmetric, and transitive.
12. A relation R is an **equivalence relation** if and only if R is reflexive, symmetric, and transitive.

Reflexivity, symmetry and transitivity are so common conditions and of such an importance that you should be able to formulate these conditions by heart!

Usually, when a relation is said to have one of the above properties this is not meant to hold just relative to some specific domain, situation, or example. For example, suppose we are looking at a specific set of four people that, it happens just so, all love themselves. We surely wouldn't say that this particular, possibly exceptional case shows that the relation 'x loves y' is reflexive in general. Likewise, even though it is often the case that when x is a friend of y and y is a friend of z then x is also a friend of z this need not be so in general. You probably have meet a friend of one of your friends once in your life that you wouldn't consider your friend. However, not only mathematical but also 'everyday' relations sometimes have one or more of the above properties in general, i.e. regardless of what domain we're considering.

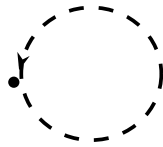
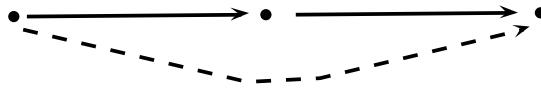
✱ **Example 5** Here are some examples of relations and their properties:

- (1.16) If in general the Neocortex is part of the brain and the brain is part of human beings, then in general the Neocortex is part of human beings. This example suggests that the part-of relation is transitive.⁸
- (1.17) If x is a relative of y , then y is also a relative of x , hence being the relative of someone is symmetric.
- (1.18) If x is a descendant of y and y is a descendant of z then x is also a descendant of z . Being a descendant of someone is transitive.
- (1.19) If x meets y then y meets x . Thus, meeting someone is symmetric.
- (1.20) If x is the father of y then y is not the father of x . Being the father of someone is antisymmetric.

It is sometimes helpful to visualize properties of relations by drawing diagrams. Figure 1.6 depicts some common properties.

⁸ This claim has been disputed from time to time. Can you think of a counter-example?

transitivity



reflexivity



symmetry

Euclideaness

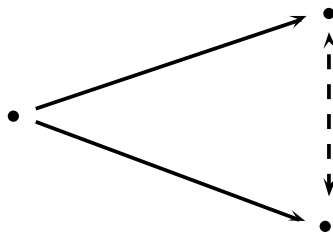




Figure 1.6: Common properties of relations


1.2.5 Exercises

 **Exercise 9** Consider a domain containing four individuals a, b, c, d , where a is a constant for Afonso, b denotes Maria's book, c stands for Afonso's car, and d stands for Maria.

- a. Maria likes Afonso but doesn't like his car. Afonso likes Maria and likes her book. Define a binary relation x likes y that satisfies these constraints. Make additionally sure that inanimate objects cannot like anything.
- b. Define a relation x belongs to y to analyze the possessives 'Afonso's car' and 'Maria's book'.

 **Exercise 10** Determine which of the following relations are (in general) transitive:

- a. *place x can be reached via Lisbon public transportation from place y*
- b. *x is greater than y*
- c. *x knows y*
- d. *x is identical to y*
- e. *x is similar to y*
- f. *x is the mother of y*
- g. $\{\langle x, x \rangle \mid x \in D\}$ for any domain D

 **Exercise 11** Check which of the following relations are reflexive, symmetric, antisymmetric, Euclidean, and transitive respectively:

- a. *x is a cousin of y*
- b. *x has the same hair color as y*
- c. *x is better than y*
- d. *x is bigger than y*
- e. *x is a multiple of y ($x, y \in \mathbb{N}$)*
- f. *x has never heard of y*
- g. *x has had a conversation with y*

 **Exercise 12** List five different equivalence relations.

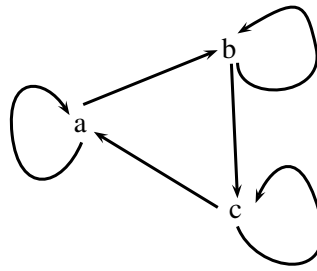


Figure 1.7: Graphical representation of a relation between three objects a, b, c .

✎ **Exercise 13** In simple models preferences are sometimes considered a total preorder. Suppose that an agent, say John, only has preferences between three alternatives a, b, c .

- Can John's preference relation contain cycles?
- Suppose Figure 1.7 depicts John's preferences. Is this a preference relation in the above sense?

1.3 Functions

1.3.1 Characterization of a Function

Function. A function maps the elements from one set, the domain of the function, to elements of another set, the codomain or image of the function, such that an element in the domain is never mapped to more than one element of the codomain. A function is partial if some elements in the domain are not mapped to anything and a function is total if all elements of the domain are mapped to an element in the codomain. There may also be elements in the codomain to which no element of the domain is mapped. Figures 1.9 and 1.10 illustrate the difference between a function and a binary relation. Figure 1.8 depicts a function as a mapping from its domain to its codomain.

When a total function is considered and elements of the codomain are ignored when nothing maps to them (i.e. the function is surjective, to be explained below), then a relation suffices to describe the function. The set of ordered pairs representing that relation may not contain any two ordered pairs with different second elements whose first element is the same. For example $\langle a, b \rangle$ and $\langle a, c \rangle$ would violate this condition. A more general way to describe a function is to define it as an order triple $\langle D, C, R \rangle$, where D is a set representing the

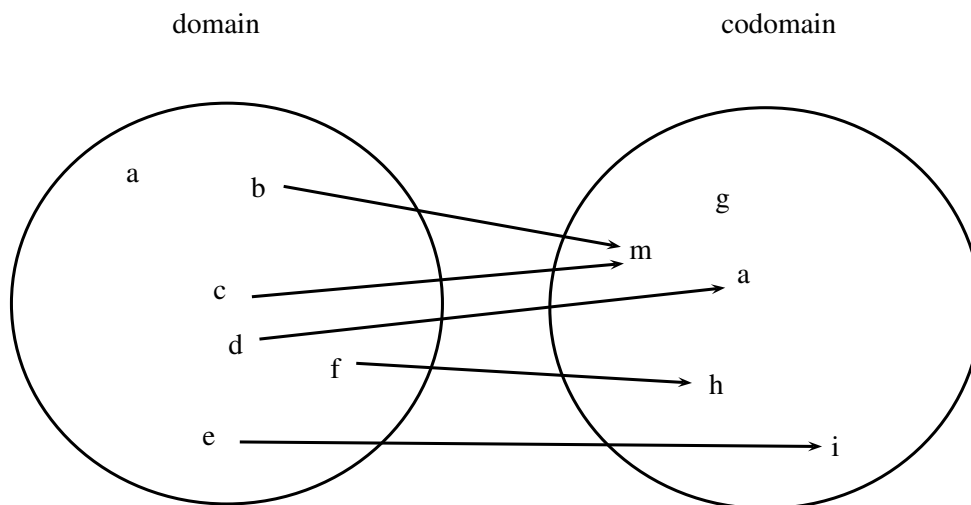


Figure 1.8: A function maps elements from the domain *once* to an element in the codomain.

domain, C is a set representing the codomain, and R is a relation consisting of ordered pairs $\langle a, b \rangle$ where $a \in D$ and $b \in C$ and additionally R satisfies the before mentioned condition that it may not contain two distinct pairs with the same first element. The key point to remember is: Two different elements of the function domain may be mapped to the same element in the codomain, but it is not allowed to map one and the same element of the domain to different elements of the codomain. In the latter case, we would only have a relation but not a function.

☞ **Note 4 (Specifying Functions.)** It is common to use small letters as function names. It is also common to write $f : A \rightarrow B$ to specify that f is a function from domain A to codomain B . There are then many ways to actually define a function and you probably know most of them from school mathematics. Only a few things that are important shall be noted here.

First, the symbol \mapsto is frequently used to specify the mapping of one particular element of the domain to an element in the codomain. Thus, for example $a \mapsto g$ means that the particular element a is mapped to a particular g . So it is better not to use \mapsto instead of \rightarrow in the notation $f : A \rightarrow B$, even though, as we will see later, \rightarrow is also commonly used for the truth-functional conditional in logical language.

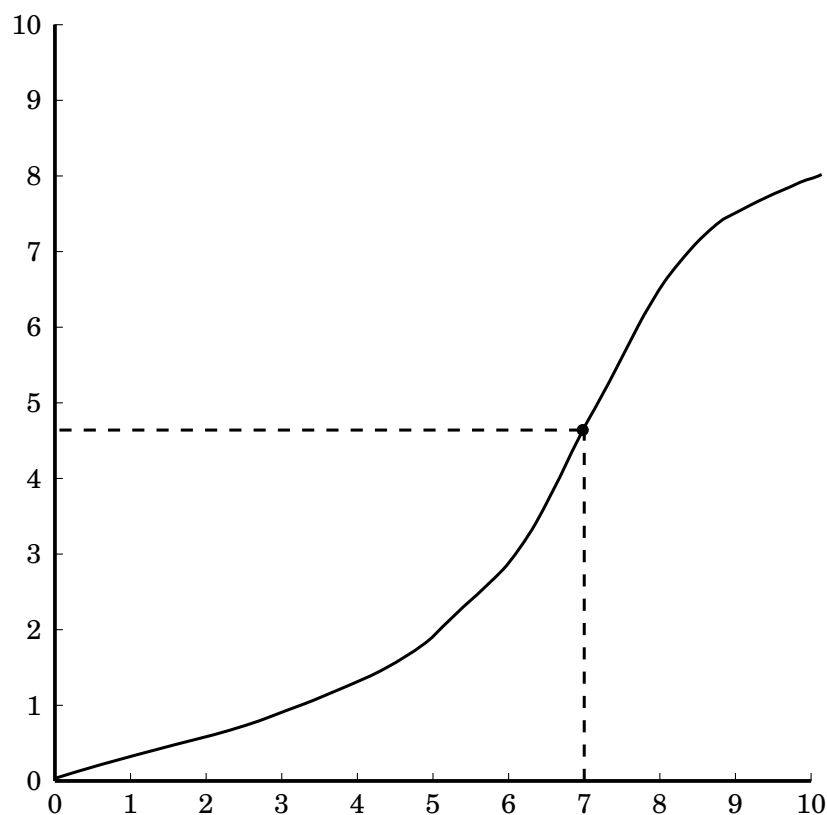


Figure 1.9: A function: Elements of the domain are mapped to exactly one element in the codomain. For example $f(7) \approx 4.7$.

Secondly, case distinctions may be used when defining a function. For example,

$$f(x) = \begin{cases} 3 & \text{if } x = 4, \\ x & \text{otherwise} \end{cases}$$

is a correct definition although it might disturb someone's sense of mathematical beauty – rightly so in this particular case, but in many applications in logic and linguistics case distinctions are unavoidable.

Third, a special notation intrinsically tied to logic is that of λ -calculus. Going back to work by Alonzo Church according to this notation $(\lambda x.Px)$ is a function that 'takes an x ' and applies P to it. For example, assuming that a is a constant for the same kind of entities as the variable x , $(\lambda x.Px)a$ reduces to Pa , which when P is an ordinary predicate might for example

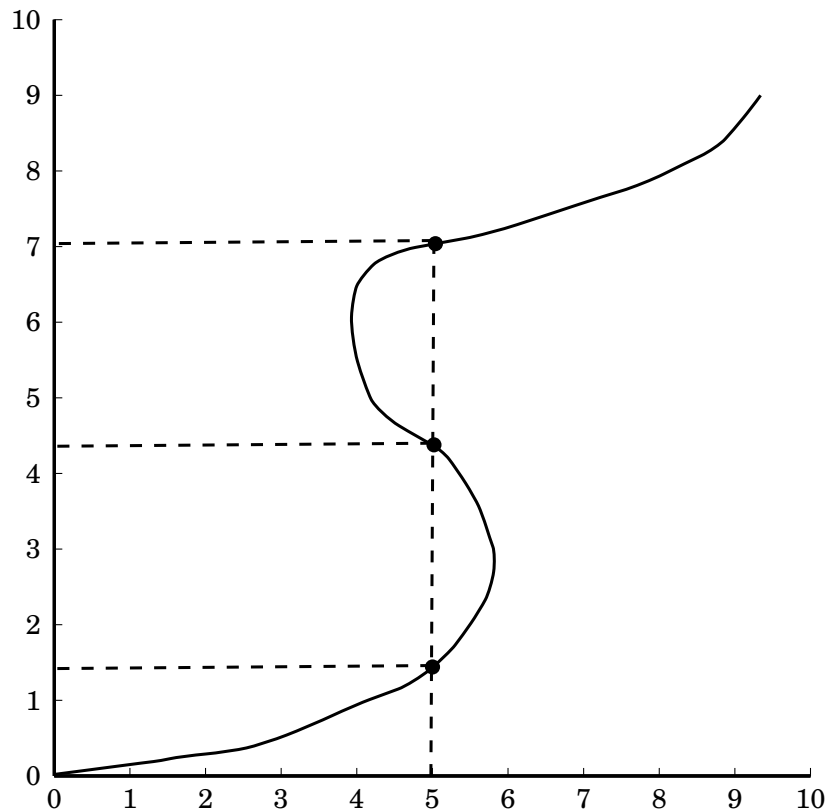


Figure 1.10: A binary relation: Elements of the first set are mapped to one or more elements of the second set. For example $R(5, 1.5)$, $R(5, 4.3)$, and $R(5, 7.04)$.

yield 1 or 0 when being evaluated. We will see in chapter 4 how this works in detail. For the time being, it suffices to be aware that $\lambda x.Px$ informally stands for a function that consumes entities of the sort of x and applies whatever there is to apply to them in its body. To give another example, $(\lambda x.x+2)3$ would yield $3+2$ via syntactic reduction rules, and interpreting $+$ and the numbers would result in 5. So in this case the meaning of $(\lambda x.x+2)$ is the function $f(x) = x + 2$.

1.3.2 Properties of Functions

Common Properties of Functions. In the following definitions $f : A \rightarrow B$.

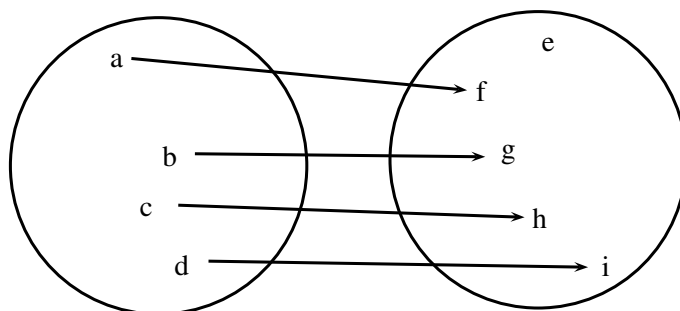


Figure 1.11: A function that is injective and not surjective.

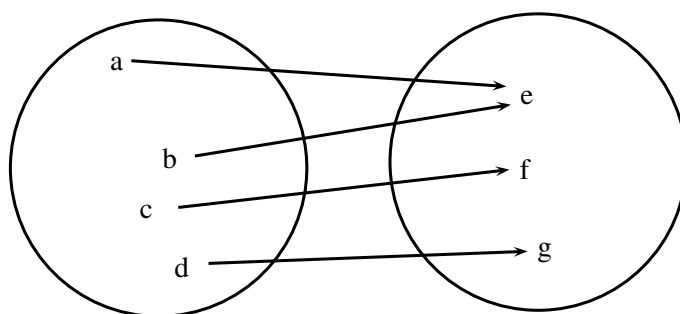


Figure 1.12: A function that is surjective and not injective.

1. A function is **surjective** or *onto* if all members of the codomain are mapped to from one or more members of the domain. See Figure 1.12.
 $\forall y \in B \exists x \in A [f(x) = y]$
2. A function is **injective** or *one-on-one* if no two or more elements of the domain are mapped to the same element in the codomain. See Figure 1.11.
 $\forall x, y \in A [(f(x) = f(y)) \rightarrow x = y]$
3. A function is **bijective** or *one-on-one and onto* if it is surjective and injective. See Figure 1.13.

Figure 1.14 depicts a function that is neither surjective nor injective.⁹

1.3.3 Further Notions

⁹ Many thanks to Ana Sofia Rocha for having pointed out a mistake in an earlier version of this picture.

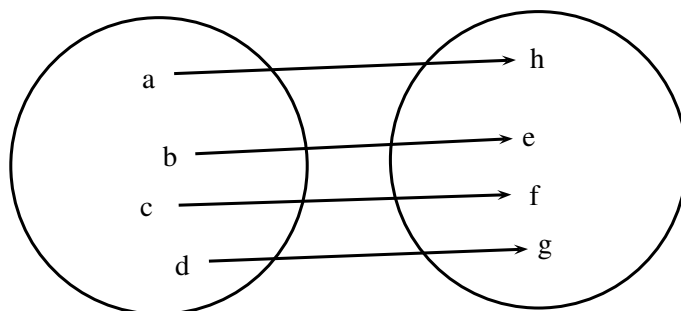


Figure 1.13: A bijective function.

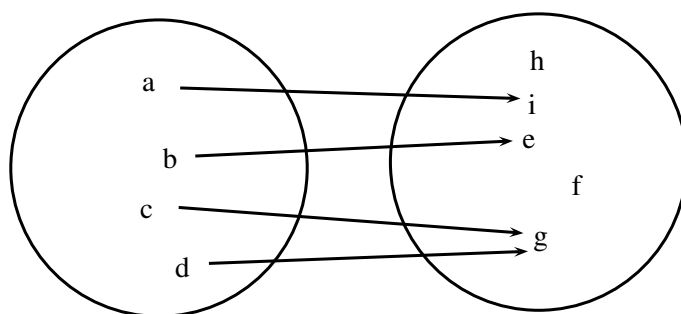


Figure 1.14: A function that is neither surjective nor injective.

Inverse Function. Let $f : A \rightarrow B$ be injective. The inverse function f^{-1} is that function $f^{-1} : B \rightarrow A$ such that $f^{-1}(b) = a$ if and only if $f(a) = b$.


Notice that f^{-1} is guaranteed to be a unique function, because f is injective. If f was not injective, then the reverse mapping would only result in a relation. If f is a bijection, then f^{-1} is also a bijection, hence the name bijection. Bear in mind, though, that from the fact that a function is injective (or bijective) alone nothing can be concluded about how easy it is to construct or compute the inverse function – unless we are speaking about functions based on finite sets whose members can be listed easily.

Characteristic Function / Indicator Function. A characteristic function or indicator function is a function $1_A : B \rightarrow \{1, 0\}$ ($A \subseteq B$) defined as


$$1_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

In other words, the characteristic function of a set A yields true if its argument is an element of A , false otherwise. We use 1,0 for truth and falsity here and will continue to do so in the subsequent sections.


1.3.4 Exercises

 **Exercise 14** Determine whether the following functions are total, partial, surjective, injective, or bijective respectively, where Dom is the domain and Cod the codomain of the function:¹⁰

- | | |
|---|---|
| a. $f(x) = x^2 \quad (x \in \mathbb{N})$ | f. $f : \{a, b, c\} \rightarrow \{a, b, c\}$ s.t. $f(x) = x$. |
| b. $f(x) = 2x - 1 \quad (x \in \mathbb{N})$ | g. $f : \{a, b, c\} \rightarrow \{a, b, c, d\}$ s.t. $f = \{(a, b), (b, c), (c, d)\}$ |
| c. $f(x) = 1/x^2 \quad (x \in \mathbb{N})$ | h. Dom = \mathbb{R} , Cod = $\{1, 0\}$,
$f(x) = \begin{cases} 1 & \text{if } x > 2 \\ 0 & \text{otherwise} \end{cases}$ |
| d. Dom = $\{a, b, c\}$, Cod = $\{1, 0\}$,
$f = \{(a, 1), (b, 1), (c, 1)\}$ | i. Dom = $\{a, b, c, d\} = \text{Cod}$,
$f = \langle \text{Dom}, \text{Cod}, \{(a, c), (b, c), (c, d)\} \rangle$ |
| e. Dom = $\{a, b, c\}$, Cod = Dom,
$f = \{(a, a), (c, c), (b, c)\}$ | |

 **Exercise 15** Specify the inverse function if there is one:


- | | |
|--|----------------------|
| a. $f(x) = x/2$ | c. $f(x) = x$ |
| b. $f(x) = x^2$ | d. $f(x) = \sqrt{x}$ |
| e. $f(x) = \begin{cases} 1.0 & \text{if } x < 0, \\ 0.5 & \text{if } x = 0, \\ 0 & \text{if } x > 0 \end{cases}$ | |
| e. Cod = $\{\text{Maria}, \text{Thomas}, \text{Peter}\}$, Dom = $\{\text{Ana}, \text{Klaus}, \text{Teresa}\}$,
$f = \{(\text{Ana}, \text{Thomas}), (\text{Teresa}, \text{Peter}), (\text{Klaus}, \text{Maria})\}$ | |


 **Exercise 16** Determine which of the following relations is also a function and specify the inverse function if there is one:


- the relation between all Turkish proper names and their referents
- the relation between all Portuguese sentences to their possible translations into English
- the relation between all passport owners to the number of their passport

¹⁰ Different notations are used deliberately.


- d. the relation between all grammatically-well formed English sentences to their meanings
- e. the relation between all dog owners and their dogs

 **Exercise 17** Let $D = \{\text{Ana, Pedro, Mustafa, Joe, Lisa}\}$, where everybody knows himself, Ana knows Pedro and Mustafa, Pedro knows Ana, Mustafa, and Joe, Mustafa knows Joe and Lisa, and Lisa knows everyone. Specify a function $f(x, y)$ with domain D that yields true if x knows y and false otherwise.

 **Exercise 18** If $f(x) = x$ then x is called a *fixed point* of f . Specify two functions that have a fixed point and their fixed point. (Do not use Google to look them up!)

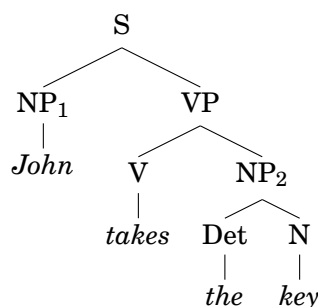
 **Exercise 19** Specify the characteristic function of the following sets:

- a. the set of all native speakers of German
- b. $A = \{1, 0, -1\}$
- c. the set of any ordered pair of two persons x and y , where x says 'Hi!' to y
- d. the set of all raining events
- e. the set $A = \{x \mid x \in \{1, 2, 3, 4, 5\}\}$ ($A \subset B$), where $B = \{1, 2, 3, 4, 5, 6, 7\}$
- f. the union of A and B , where $A = \{a, b, c, d\}$ and $B = \{a, b, d, f\}$
- g. the set of ordered quintuples $\langle x, y, z_1, z_2, t \rangle$ such that x buys y from z_1 at price z_2 at time t
- h. the set of all non-black ravens

 **Exercise 20** Determine which of the following relations can also be regarded as functions:

- a. $\{\langle a, a \rangle, \langle b, b \rangle, \langle c, a \rangle, \langle d, 1 \rangle\}$
- b. $\{\langle a, a \rangle, \langle b, a \rangle, \langle c, a \rangle, \langle d, 1 \rangle\}$
- c. $\{\langle a, a \rangle, \langle b, b \rangle, \langle a, b \rangle, \langle d, 1 \rangle\}$
- d. $\{\langle b, b \rangle, \langle a, a \rangle, \langle d, 1 \rangle, \langle c, a \rangle, \langle e, f \rangle\}$
- e. f^{-1} , where $f(x) = \frac{x^2}{x}$ ($x > 0$)
- f. $R = \{\langle x, y \rangle \mid x, y \in \mathbb{N} \text{ and } x > y\}$
- g. to know someone
- h. the relation between a sales item and its price
- i. the relation between the number of letters printed in a book and the physical thickness of the book

✎ **Exercise 21** In natural language syntax, the notion of c-command in a syntactic derivation tree is defined as follows. A node A dominates a node B if A is higher than B in the tree and you can draw a line from A to B by only going downwards.¹¹ Node A c-commands node B if and only if (i) A does not dominate B , (ii) B does not dominate A , and (iii) every node that dominates A , also dominates B . For example, in the following tree NP_1 c-commands V and NP_2 :



- Can ' x c-commands y ' be represented by a function?
- Can ' x is c-commanded by y ' be represented by a function?

1.4 Literature

There are many introductions to set theory and discrete mathematics. Just about any text you find appealing will do. I have for example used the following ones for self-study and preparations:

- Kenneth H. Rosen (2007). *Discrete Mathematics and its Applications*. McGraw-Hill.
- Barbara H. Partee, Alice ter Meulen, and Robert E. Wall (1990). *Mathematical Methods in Linguistics*. Springer.

Generalized quantifiers have first been investigated in detail from a formal point of view by A. Mostowski *On a generalization of quantifiers*, Fund. Math. Vol. 44, pp. 12-36. Partee/ter Meulen/Wall provide a good and accessible overview. Another accessible overview can be found in the following general introduction to semantics:

- Irene Heim and Angelika Kratzer (1998). *Semantics in Generative Grammar*. Blackwell.

¹¹ Giving a more precise definition is more complicated and best done recursively.

Frege's articles are a must-read for anyone interested in meaning. For linguistics his most important articles are:

- Gottlob Frege (1892). Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, 25-50.
- Gottlob Frege (1918-1919). Der Gedanke. Beiträge zur Philosophie des deutschen Idealismus 2 (1918-1919), 1918, 58-77.

English translations are ubiquitous, see for example Peter Geach and Max Black (1980). *Translations from the philosophical Writings of Gottlob Frege*. Blackwell.

Propositional Logic

In this chapter we take a look at propositional logic (often abbreviated *PC* for ‘propositional calculus’). In propositional logic there are no quantifiers: It only deals with constants that stand for entire natural language sentences and the ways these constants may be combined to form more complex expressions. For example, let p and q stand for sentences; then we can express p and q by the formula $p \wedge q$ in the language of propositional logic. It is clear that propositional logic alone is not of much use for linguistic theorizing. However, it is the basis for more expressive logical languages in which predicates, relations, variables and constants for objects, and ways to quantify over variables are available. Everything that can be said about propositional logic will transfer neatly to those more expressive languages.

2.1 Syntax of Propositional Logic

We start by defining a formal language, i.e. a set of strings that characterizes all well-formed expressions of that language, and call this language *PC*. Later on, the goal will be to approximate the syntax of the formal language as nearly as possible to that of a natural language or at least provide an approximation that allows us to obtain good semantic representations from an underlying syntactic theory and the specification of a lexicon for a natural language. Since propositional logic is only concerned about combinations between sentences without taking into account quantifiers, we will not even come close to this goal in this section.

2.1.1 Basic Expressions

Propositional Constants. We use p, q, r and their indexed variants such as $p', q'', r''', p_1, q_2,$ and r_{99} as propositional constants. Let L_C be the set containing

them. We additionally assume that L_C contains to special constants \top called Verum and \perp called Falsum.¹

Connectives. Connectives are used to combine propositional constants into more complex expressions. Let $L_B := \{\vee, \wedge, \rightarrow, \leftrightarrow, \uparrow, \downarrow, \dot{\vee}\}$ be the set of binary connectives for PC and $J_N := \{\neg\}$ be the set of unary truth-functions containing only the negation symbol \neg . We will soon see what exactly these mean. For now, here is a list of their common names and some alternative notation:

Conjunction. \wedge , $\&$, *AND*

Disjunction. \vee , $;$, *OR*, $|$

more precisely called *inclusive disjunction*, very rarely also called *adjunction*

Negation. \neg , \sim , $-$, *NOT*, *NEG*, \bar{p} (overlining the expression to be negated)

more precisely called *truth-functional negation*, very rarely also called *outer negation*

Conditional. \rightarrow , \supset , \Rightarrow , *IMPLIES*, *IF...THEN*

sometimes also called *implication*, *material implication*, rarely also called *subjunction*

Biconditional. \leftrightarrow , \equiv , *IFF*, \Leftrightarrow

sometimes also called *equivalence*, *material equivalence*, *if and only if*, *iff.*, rarely also called *bisubjunction*, *biimplication*

Exclusive Disjunction. $\dot{\vee}$, *XOR*, $|$

sometimes also called *exclusive OR*

Sheffer Stroke. \uparrow , *NAND*, $|$, *i* and similar symbols

Peirce Stroke. \downarrow , *NOR*, \dagger , $!$ and similar symbols

sometimes also called *Quine dagger* or referred to as one of the Sheffer strokes

Parentheses are used for making scope distinctions; to enhance readability we will allow [and] as notational variants of (and).

¹ The symbols \top and \perp are sometimes used for other purposes. Don't bother too much with the choice of symbols.

✧ **Remark 3 (Arrow or horseshoe?)** The conditional is of particular importance in logic and many excellent logicians use the horseshoe \supset instead of the right arrow for the conditional. In a sense this notation is a bit misleading; there is a close connection between the conditional \rightarrow and the subset relation \subseteq of set theory and so the horseshoe is pointing into the wrong direction. On the other hand, the horseshoe brings luck and so the final word has not yet been spoken on this issue of uttermost importance.

📖 **History 4 (A Well-Known Anecdote)** This is not really related to the use of ' \supset ' in logic. A visitor is said to once have asked Niels Bohr, the famous physicist, whether he really believed the horseshoe above the door of his house would bring him luck. Bohr replied: "Of course not... but I am told it works even if you don't believe in it."²

2.1.2 Well-Formed Formulas

Given the basic entities defined in the previous section, the syntax of our formal language may be defined in various ways. A very common one is by giving a recursive definition like the following one.

(Syn1) Well-Formed Formula. The set L_S of well-formed formulas of PC is defined as follows:

1. If $\phi \in L_C$ then $\phi \in L_S$.
2. If $\phi \in L_S$ and $\tau \in L_N$, then $\tau\phi \in L_S$.
3. If $\phi, \psi \in L_S$ and $\circ \in L_B$ then $(\phi \circ \psi) \in L_S$.
4. Nothing else is in L_S .

Let's abbreviate *well-formed formula* as *wff* and go through the definition. The Greek letters ϕ and ψ are used as meta-variables for propositional constants, ν stands for a unary junctor – which in our case is only \neg – and \circ stands for any of the binary junctors such as \wedge , \vee , or \rightarrow . The definition first states that any propositional constant alone is a wff. Then it states that when we put the

² It is disputed whether this dictum can really be attributed to Niels Bohr; it might in fact have been made by one of his neighbors and he liked the reply.

negation sign \neg in front of a wff, the result will be a wff and that any two wffs can be combined by an infix binary junctor and when wrapped into parentheses will result in a wff. So the definition defines for example the following strings as wffs: p , r_{78} , $(p \rightarrow q)$, $((p \wedge q) \rightarrow p)$, $\neg(\neg p \wedge \neg q)$, $((p \wedge q) \vee (\neg p \wedge \neg q)) \leftrightarrow (p \leftrightarrow q)$. These are just a few examples and it is easy to see that $|L_S| = \aleph_0$. We can enumerate all wffs by starting with the simplest ones and combine them systematically to produce more complex wffs but there is no finite limit to the size of a wff; the above definition produces countably infinitely many wffs.³

It is common to leave out outer parentheses to make wffs more readable and we will do that, too. So instead of $(p \wedge q)$ we may write $p \wedge q$ and instead of $((p \wedge q) \rightarrow r)$ we may write $(p \wedge q) \rightarrow r$. It is also common to define an order of precedence for the connectives, often one according to which \wedge and \vee bind stronger than \rightarrow and \leftrightarrow . According to such conventions, it might for example be allowed to write $p \wedge q \rightarrow r$ for $(p \wedge q) \rightarrow r$. This frequently confuses beginners and so we will not adopt precedence rules in this chapter. If you encounter a text with precedence rules and are in doubt about the right grouping of the symbols, apply the following general rule:

In case of doubt, refer to the definition!

Although full precedence rules are not used in this chapter, we will allow a notational convention in order to make formulas more readable. When we have several subsequent applications of conjunction we may leave out parentheses except the outermost if they are necessary. For example, instead of $(p \wedge (q \wedge r)) \rightarrow p$ we may write $(p \wedge q \wedge r) \rightarrow p$. This is admissible because, as we shall soon be able to prove, $(\phi_1 \wedge (\phi_2 \wedge \phi_3)) \leftrightarrow ((\phi_1 \wedge \phi_2) \wedge \phi_3)$. Not only that, as it will become apparent when a precise semantics for \wedge has been given, in any sequence of conjunctions the arguments can be permuted arbitrarily without having any effect on the truth value of the formula as a whole. Since so far only syntax void of any meaning is available these claims have to be taken for granted for the time being.⁴

As you probably know or might easily imagine, there are many other ways to define the syntax of a formal language. Because of its importance one more method deserves mentioning. Some logicians and many computer scientists sometimes use a very compact notation for defining wffs that looks like this:

$$S := (\phi \wedge \psi) \mid (\phi \vee \psi) \mid (\phi \rightarrow \psi) \mid (\phi \leftrightarrow \psi) \mid \neg \phi$$

or, slightly more correctly,

³ A diligent logician would prove this claim by induction on the size of formulas, but this level of detail is beyond the scope of this introductory course.

⁴ When asked about some issue early during his talk, a logician whose name I cannot remember once answered at a conference: “I cannot answer this question, because I have only introduced the syntax by now.” (It was a joke.)

$$S := (S_1 \wedge S_2) | (S_1 \vee S_2) | (S_1 \rightarrow S_2) | (S_1 \leftrightarrow S_2) | \neg S$$

These are essentially just concise variants of the definition given above.

Main Junctor. The main junctor or connective in a formula is the occurrence of a connective at the topmost priority level of that formula. For example, \rightarrow is the main junctor in $p \rightarrow (q \wedge r)$ and \uparrow is the main junctor in $(p \downarrow q) \uparrow (p \downarrow q)$. Of course, a formula can contain more than one occurrence of a connective. Intuitively it is not very hard to understand what is meant by ‘topmost priority level.’ If we want to make the notion more precise, it is best to define it simultaneously with the definition of a wff:

(Syn2) Well-formed Formula and Main Junctor.

1. If $\phi \in L_C$ then $\phi \in L_S$ and ϕ has no main junctor.
2. If $\phi \in L_S$ and $\nu \in L_N$, then $\nu\phi \in L_S$ and the main junctor of ϕ is ν .
3. If $\phi, \psi \in L_S$ and $\circ \in L_B$ then $(\phi \circ \psi) \in L_S$ and the main junctor of ϕ is \circ .
4. Nothing else is in L_S .

The idea is here that previous determinations of the main junctor are overruled by the very last one. So for example, p and q have no main junctor, the main junctor of $p \wedge q$ is \wedge , the main junctor of $p \vee q$ is \vee , and finally the main junctor of $(p \wedge q) \rightarrow (p \vee q)$ is \rightarrow .

📖 History 5 (Polish and Reverse Polish Notation) Before fancy typesetting was available, authors often used some more compact notation without any parentheses. Famous Polish logician Jan Łukasiewicz introduced Polish notation depicted in table 2.1. For example, the propositional formula $(p \wedge q) \leftrightarrow \neg(\neg p \vee \neg q)$ for one of DeMorgan’s laws is written $EKpqNANpNq$ in Polish notation.

In reverse Polish notation, the arguments are written first and then the functor last. So for instance $(p \wedge q)$ is written pqK . This notation is still used for some scientific calculators and programming languages like Forth.

2.1.3 Exercises

📖 Exercise 22 Determine which of the following formulas are well-formed according to definition S1 and which aren’t. (Outer parentheses may have been left out.)

Common Notation	Polish Notation
$\neg\phi$	$N\phi$
$\phi \wedge \psi$	$K\phi\psi$
$\phi \vee \psi$	$A\phi\psi$
$\phi \rightarrow \psi$	$C\phi\psi$
$\phi \leftrightarrow \psi$	$E\phi\psi$
$\phi \uparrow \psi$	$D\phi\psi$

Table 2.1: Polish notation for propositional logic.

- | | |
|---|---|
| a. $((p \rightarrow q) \wedge (r \leftrightarrow q)) \rightarrow p$ | g. $p \rightarrow (q \rightarrow (p \rightarrow q))$ |
| b. $p \wedge q \vee r$ | h. $p \uparrow (q \dot{\vee} \neg(r_3 \downarrow r_3))$ |
| c. $((p \leftrightarrow q)(r \wedge p))$ | i. $(p \wedge (q \rightarrow (q \leftrightarrow p)))$ |
| d. $((\neg p) \wedge (q \rightarrow r_2))$ | j. $((p \vee q) \wedge r) \vee q_2 \leftrightarrow p'$ |
| e. $(a \wedge (b \vee c))$ | k. $p \leftrightarrow p$ |
| f. $(P(x) \leftrightarrow Q(x))$ | l. $(q \neg \leftrightarrow (p \vee r))$ |

✎ **Exercise 23** Underline the main junctor of the following wffs if there is one:

- $((p \wedge q) \vee (r \wedge (q \vee r)))$
- q
- $\neg(p \leftrightarrow q)$
- $(p \wedge q) \uparrow (\neg p \wedge \neg q)$
- $(p \wedge q) \rightarrow p$
- $((p \rightarrow q) \vee (q \rightarrow r)) \rightarrow ((p \vee \neg q) \vee (\neg p \vee q))$
- $\neg(\neg p \vee \neg q) \leftrightarrow (p \wedge q)$
- $\neg p$

2.2 Semantics of Propositional Logic

So far we have only defined a way of notating formulas of a concise formal language. These are supposed to stand for sentences, sentence negation and combinations between sentences. So we now have to define how well-formed

formulas, i.e. sentences of our formal language PC , are to be translated. We will now specify the semantics of PC by giving rules that assign a meaning to each of the infinitely many well-formed formulas.

A crucial thing to note about this semantics is that is based on a strong but powerful idealization: The meaning of a complete sentence, here simply represented by a propositional constant, is considered to be either truth or falsity and nothing else. This might sound inadequate at first glance, but we will see that it is amazingly powerful. In any case, propositional logic will turn out to be the basis of many more powerful semantic representations. We begin by specifying a model for PC and subsequently define what it means for a formula to be true in a model.

2.2.1 Models and Truth in a Model

Model. A model M for propositional logic consists of an interpretation function $I : L_C \rightarrow \{1, 0\}$ for propositional constants.

Truth in a Model. We define truth in a model, writing $M \models \phi$ for the fact that a well-formed formula ϕ is true in model M . (And we write $M \not\models \phi$ for the fact that it is not the case that $M \models \phi$.) \models is defined recursively as follows:

- $M \models \phi$ iff. $I(\phi) = 1$ (for $\phi \in L_C$).
- $M \models \neg\phi$ iff. $M \not\models \phi$.
- $M \models (\phi \wedge \psi)$ iff. $M \models \phi$ and $M \models \psi$.
- $M \models (\phi \vee \psi)$ iff. $M \models \phi$ or $M \models \psi$ (or both).
- $M \models (\phi \dot{\vee} \psi)$ iff. either $M \models \phi$ or $M \models \psi$ (but not both).
- $M \models (\phi \rightarrow \psi)$ iff. $M \not\models \phi$ or $M \models \psi$.
- $M \models (\phi \leftrightarrow \psi)$ iff. either $M \models \phi$ and $M \models \psi$, or both $M \not\models \phi$ and $M \not\models \psi$.
- $M \models (\phi \uparrow \psi)$ iff. it is not the case that: $M \models \phi$ and $M \models \psi$.
- $M \models (\phi \downarrow \psi)$ iff. it is not the case that: $M \models \phi$ or $M \models \psi$.
- $M \models \top$ is always true.
- $M \models \perp$ is always false.

☞ **Note 5 (Object versus Meta-Language)** The above denotational semantics provides an interpretation to the well-formed formulas that have been defined in the previous section. When talking about logical systems the object-language has to be distinguished from the meta-language. In case of *PC* the object language is the set of well-formed formulas L_S defined in the previous section, or, if we take a look at the formal language plus its interpretation, L_s in combination with the above denotational semantics. The meta-language, on the other hand, is English with a bit of mathematical vocabulary. It is also possible to define a formal language on the basis of another formal language.

2.2.2 Truth Tables

According to the above definition of truth in a model all connectives are interpreted as *truth-functions* taking well-formed formula as an argument and yielding a truth-value. Instead of the above semantics we can also use *truth-tables* to specify these functions. For example, truth-functional negation flips the truth-value:

Negation	
ϕ	$\neg\phi$
1	0
0	1

Here are the tables for the binary connectives:

Conjunction			Disjunction			Conditional		
ϕ	ψ	$\phi \wedge \psi$	ϕ	ψ	$\phi \vee \psi$	ϕ	ψ	$\phi \rightarrow \psi$
1	1	1	1	1	1	1	1	1
1	0	0	1	0	1	1	0	0
0	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	0	1

Biconditional			Exclusive Disjunction			Sheffer Stroke		
ϕ	ψ	$\phi \leftrightarrow \psi$	ϕ	ψ	$\phi \dot{\vee} \psi$	ϕ	ψ	$\phi \uparrow \psi$
1	1	1	1	1	0	1	1	0
1	0	0	1	0	1	1	0	1
0	1	0	0	1	1	0	1	1
0	0	1	0	0	0	0	0	1

Peirce Stroke		
ϕ	ψ	$\phi \downarrow \psi$
1	1	0
1	0	0
0	1	0
0	0	1

If you study these tables carefully, you'll likely come to the conclusion that they do not specify all possible truth functions.⁵ It is not hard to see that there are $2^2 = 4$ unary truth-functions, among which negation is just one. The others are not very interesting, though. For example, the identity function maps 1 to 1 and 0 to 0. The other two just yield false or true respectively, no matter what input they get. Likewise, there are $2^4 = 16$ binary truth-functions, but the most interesting ones are displayed above.

☞ **Note 6 (All Truth Functions)** Here is a list of all 16 binary truth functions:

p	q	①	\vee	②	\rightarrow	\uparrow	③	④	⑤	⑥	\leftrightarrow	$\dot{\vee}$	\downarrow	⑦	⑧	\wedge	⑨
1	1	1	1	1	1	0	1	1	0	0	1	0	0	0	0	1	0
1	0	1	1	1	0	1	1	0	1	0	0	1	0	0	1	0	0
0	1	1	1	0	1	1	0	1	0	1	0	1	0	1	0	0	0
0	0	1	0	1	1	1	0	0	1	1	1	0	1	0	0	0	0

The common functions are really just \wedge , \vee , \rightarrow , and \leftrightarrow . The functions marked with circled numbers are often not named. If you absolutely can't live without giving them names, here are some suggestions:

① Verum function; ② converse conditional; ③ projection of first argument; ④ projection of second argument; ⑤ negation of second argument; ⑥ negation of first argument; ⑦ converse nonconditional; ⑧ nonconditional; ⑨ Falsum function.

Don't use these names! Everybody will understand the truth table but not many people will associate anything useful with the the names for ① - ⑨.

Notice that according to the definition of truth in a model and the above tables the specific semantic content or meaning of a sentence is disregarded and its truth value in a model is taken into account by using the valuation function to interpret the corresponding propositional constant. Take $D_t = \{1, 0\}$ as the

⁵ Or, you don't study them carefully and instead skim through Wikipedia. However, from just looking things up you do not learn how to think and solve problems on your own.

set of truth-values. The interpretation of a unary connectives is a function $f : D_t \rightarrow D_t$ and interpretation of a binary connective is a function $f : (D_t \times D_t) \rightarrow D_t$. Hence, it is correct to say that unary connectives are interpreted as a unary truth-function from truth-values to truth-values and binary connectives are interpreted as a binary function from the set of all ordered pairs of truth-values to truth-values. However, the distinction between the syntactic entity, i.e. a connective, and its interpretation is often mixed up and in a more loose way of talking it is quite common to speak about truth-functions for both the connectives as syntactic entities and their interpretation.

As it turns out at a closer look, the truth-functions are mostly interdefinable. Take for example the conditional $p \rightarrow q$, which is only false if the antecedent p is true and the succedent q is false and compare it with the following expression: $\neg p \vee q$. They have exactly the same truth-values, irrespective of what the actual values of p and q are. We prove this by constructing a table that takes into account all 4 possible combinations of values of p and q :

ϕ	\rightarrow	ψ	\neg	ϕ	\vee	ψ
1	1	1	0	1	1	1
1	0	0	0	1	0	0
0	1	1	1	0	1	1
0	1	0	1	0	1	0

As you can see the main junctors of both formulas have exactly the same entry $\langle 1, 0, 1, 1 \rangle$. So the truth conditions of these formulas are exactly the same. This proves that \rightarrow can be defined in terms of \neg and \vee by the purely syntactic abbreviation scheme $(\phi \rightarrow \psi) := (\neg \phi \vee \psi)$. What about interdefinability in general? We capture the interdefinability of connectives in propositional logic by the notion of a base for propositional logic.

Base. A set S of unary or binary connectives is a *base for classical 2-valued propositional logic* if and only if all other unary and binary connectives can be defined by the ones in S by a syntactic abbreviation scheme.

It is not hard but also not trivial to show in the detail exactly which combinations of connectives form a base and which don't. Of course, it is very simple to show of a given set of junctors S that it is indeed a base. Just define all other connectives in terms of the ones in S and proof that the definition is correct by using the method of the truth-tables illustrated above. One clear criterion for a base is that we can express negation in it. So for example, the set $\{\wedge, \leftrightarrow\}$ is not a base, because we cannot express solely in terms of conjunction and biconditional. In contrast to this, $\{\neg, \rightarrow\}$ is a base.

☞ **Note 7 (Sheffer Strokes.)** Does any base contain at least two connectives? As it happens, the answer is No. Sometimes students are surprised to hear or find out on their own that the Sheffer and Peirce strokes are bases on their own. Let us take a look why this is so in case of Sheffer stroke \uparrow . (The corresponding line of reasoning for the Peirce stroke is quite similar.) Notice that the truth-table of \uparrow is like one for conjunction except that all result values have been ‘flipped.’ Hence $\phi \uparrow \psi$ should be equivalent to $\neg(\phi \wedge \psi)$ and indeed it is. You can check this by using the method of truth-tables. Now let’s try to define negation. Obviously, negation is a unary truth-function and so it only takes one value. To simulate this, we assume only one propositional constant in whatever definition we come up with. Let’s try $p \uparrow p$. The truth-table for this is:

ϕ	\uparrow	ϕ
1	0	1
0	1	0

That is indeed negation. If you’re not yet convinced, take a look at the following, more verbose table that proves that the formula $\neg p \leftrightarrow (p \uparrow p)$ for any p and any value of p :

p	p	\neg	p	\leftrightarrow	$(p$	\uparrow	$p)$
1	1	0	1	1	1	0	1
0	0	1	0	1	0	1	0

To show that the dyadic truth-functions are definable by \uparrow larger tables with two propositional constants and all combinations of truth values between them have to be constructed.

Once it has been shown that a set of connectives S_1 is a base, then in order to show that another combination S_2 of connectives is a base it suffices to show that all the connectives in S_1 can be defined in terms of those in S_2 . Exhaustively proving of a set of connectives (possibly just containing \uparrow or \downarrow) that it is a base is easy but tedious; we omit the proof here and merely point out that $\{\uparrow\}$, $\{\downarrow\}$, $\{\neg, \wedge\}$, $\{\neg, \vee\}$, and $\{\neg, \rightarrow\}$ are common bases. That is also the reason why the syntax and semantics of propositional logic and systems extending it often just provides syntactic and semantic rules for a base like $\{\neg, \wedge\}$ and all the other connectives are defined as syntactic abbreviations.

2.2.3 Important Notions

Validity, Tautology. A wff ϕ is *valid* if and only if it is true in any model. It is common to omit reference to any particular model and write $\models \phi$ for expressing the fact that ϕ is valid. A valid wff is also called a *tautology*.

Satisfiability. A formula ϕ is satisfiable if and only if there is a model M such that $M \models \phi$.

This definition might look weird at first glance, because it relies so much on the notion of a model. The idea is, however, simply that a satisfiable formula that is not valid may be true or false depending on the state of the world, which is represented by the valuation function of the model. Compare this with a valid formula. Models represent arbitrary interpretations of our language and in a sense define the ‘logical space’, what is logically possible, with respect to what can be expressed in the language. If ϕ is true in all models this also means that there is no model in which ϕ is false. Consequently, it is not (logically) possible for ϕ to be false. In other words, a valid formula is true for purely logical reasons. In contrast to this, when a wff that is satisfiable and not valid turns out to be true in some particular model, then it is true because of the state of affairs of the world represented by that model. A wff that is satisfiable but not valid is sometimes called *contingent*.

For example, consider p to stand for ‘O Pedro ama a Maria’ and q stand for ‘A Maria gosta do Pedro’. How do we find out whether p and q are true or false respectively? We could, for example, ask Maria and Pedro and build a model accordingly. Suppose they both agree on the respective statement. Then in our model M , $I(p) = 1$ and $I(q) = 1$. Hence, $M \models p \wedge q$. Clearly $p \wedge q$ is not valid, though, as we can easily build a model M' according to which for example $I(q) = 0$. Contrast this with the statement ‘O Pedro ama a Maria ou o Pedro não ama a Maria’. Understood literally, this statement could be translated to $p \vee \neg p$ and is clearly valid, as the following truth table proves:

p	\vee	\neg	p
1	1	0	1
0	1	1	0

✧ **Remark 4 (Dependence on Analysis)** The formula $p \vee \neg p$ is valid in propositional logic where we only whole sentences and their truth values and no quantifiers are taken into account. According to a more fine-grained reading of the sentence ‘O Pedro ama a Maria ou o Pedro não ama a Maria’ one could argue that the respective translation of this sentence into a logical language ought not be valid, because there does not seem to be any rule about Portuguese proper names that warrants that a person named ‘Pedro’ or a person named ‘Maria’ actually exist; after all, whether the bearer of

a proper name exists or not can hardly be a matter of logic alone. This is indeed a good point about proper names and has been discussed extensively in the literature on empty names and existence presuppositions. There is no way to express this view in the relatively impoverished language of propositional logic, though.

Care should be taken not to prematurely reject a theoretical tool just because it is not fully adequate for describing a certain phenomenon. As long as it can be used to adequately describe other phenomena it might have its place in the theoretical apparatus of a scientist. Ultimately it is a virtue of any good theory to be restricted to a particular problem domain and ignore other phenomena. Take for example classical mechanics. In classical, Newtonian mechanics bodies are sometimes regarded a point in space whereas we all know that any real object is spatially extended. Nevertheless, nobody would reject mechanical theories just because they don't take into account the extendedness of physical bodies. In other words, bear in mind the following deep wisdom:

**Don't expect from a hammer
to be useful for drilling holes!**

Contradiction. A wff is a *contradiction* if it is not satisfiable in any model. For example, $p \wedge \neg p$ is a contradiction as the following truth table proves:

p	\wedge	\neg	p
1	0	0	1
0	0	1	0

Consistency. Two wffs are consistent with each other if they are satisfiable conjointly, i.e. if there is a model in which both formulas are true. In other words, two wffs are consistent if they are not contradictory to each other. Likewise, a set of wffs is consistent if there is a model in which all of them are true, or, put in other terms, if the conjunction of these wffs is not a contradiction.

Notice that the term 'coherence' is sometimes used in a similar sense as consistency, but has a much broader meaning. For example you could claim that a sentence like 'If the moon is made of green cheese then 1 equals 1' is consistent, but that there is no coherent connection between the antecedent and the succedent of 'If... then' in that phrase.

Equivalence Between Formulas. Two wffs are equivalent if and only if they are satisfied by exactly the same models. In other words, two formulas ϕ and ψ are equivalent if and only if for all models M , $M \models \phi$ iff $M \models \psi$. Despite some obvious similarities this way of talking about equivalence between formulas in a logical language needs to be kept apart from the general notion of an equivalence relation. (See note 8 for more details.)

It is useful to memorize the following rules about the connection between satisfiability, validity, and contradictions:

- A valid wff is satisfiable (in any model).
- The negation of a valid wff is a contradiction.
- The negation of a contradiction is valid / is a tautology.
- The negation of a tautology is a contradiction.

☞ **Note 8 (Abbreviation vs. Equivalence vs. Biconditional)** Recall our use of the notation $:=$ to indicate syntactic abbreviations. Now that a formal language is available with a syntax and its interpretation in a model this is clearly distinct from similar and closely related symbols like $=$ or \leftrightarrow . The notation $:=$ really just indicates a syntactic abbreviation that allows you to replace the string on the left-hand side in a formula by the string on the right-hand side, as long as the syntax of the language and the parentheses are respected. For example, if $\phi \rightarrow \psi := \neg\phi \vee \psi$, the wff $(p \rightarrow q) \rightarrow p$ must be rewritten as $\neg(\neg p \vee q) \vee p$ no matter what \rightarrow , \neg , or \vee actually mean.

In contrast to this, $a = b$ stands for identity, meaning that the constants 'a' and 'b' denote the same object.⁶

Yet in contrast to this, $p \leftrightarrow q$ is a truth-function that yields 1 if either the value of both p and q is 1 or both have the value 0. Bear in mind that \leftrightarrow takes propositional constants and not constants or variables for objects, whereas the arguments of $=$ are objects in the very broad sense.

Notice the close relation between all of these notions, though. We could have defined $M \models \phi \leftrightarrow \psi$ as yielding 1 if $I(\psi) = I(\phi)$, 0 otherwise. Moreover, a formula defined by syntactic abbreviation just *is* the right-hand side and thus only gets its meaning from the meaning of the right hand side. But what about equivalence? Recall from the last chapter that any symmetric, reflexive, and transitive relation is an equivalence relation. So when we say a and b are equivalent it always means equivalent with respect to a particular equivalence criterion. Now identity is an equivalence relation between objects based on the criterion that they are really one and the

same object, i.e. $\{\langle x, x \rangle \mid x \in D\}$ for a given domain D of objects, and the biconditional is an equivalence relation based on the criterion that the formulas on each of its sides have identical truth values. For this reason, it is also common to call the truth-function \leftrightarrow equivalence or speak of equivalence between formulas when both of them are satisfied in exactly the same models.

Theory, Intended Model. A set of sentences (wffs) of a logical language is often called a *theory*. Usually this is meant in such a way that the wffs in the set are interpreted in *intended models*; these are the models in which all the non-logical constants have their intended meaning, as it is characterized by paraphrases. Of course, regarding a set of formulas a theory is an idealization.

The Method of Truth Tables. It is useful to memorize the following rules for the use of truth tables:

- If the column of the main junctor of a complete truth table has the value 1 for any combination of values of the propositional constants, then the whole wff is valid.
- If the column of main junctor of a complete truth table has the value 0 for any combination of values of the propositional constants, then the whole wff is a contradiction.
- If the column of the main junctor of a complete truth table contains both 0 and 1's then the wff is satisfiable but not valid.
- If the columns of the main junctors of two complete truth tables for two formulas ϕ and ψ coincide, then ϕ and ψ are equivalent.

2.3 Proof Theory

In this section we will look at a *proof theory* for propositional logic. Proof theories are more or less mechanical methods for testing whether a wff is valid or not. We have already seen the method of truth tables that indeed presents a limited

⁶ It is not that simple. In 'Über Sinn und Bedeutung' Gottlob Frege considers this so called *meta-linguistic* view of identity counter-intuitive, because symbols are arbitrary, and instead suggested that the senses (or, in another parlance, meanings) of a and b determine the same object. Which notion of identity is the right one is still a controversial issue in philosophy and the difference in opinion matters a lot for linguistics, because Neo-Fregean views about meaning have given rise to intensional semantics. I'll leave this issue open for now.

form of a proof theory, but truth tables cannot be regarded as a practical proof theory in general. Why not? The tables we have seen so far were based on wffs with two distinct propositional constants and had four rows. A table for 3 propositional constants has 8 rows. To construct it, take a table for two propositional constants, duplicate the rows, and add the rows $\langle 1, 1, 1, 1, 0, 0, 0, 0 \rangle$ for the third propositional constant. In general, a table for n propositional constants has 2^n rows. Thus, a table for four variables has 16 rows, and a table for five variables has 64 rows. That's about the maximum number of rows that are practical without the aid of a machine. Since the grow in size is exponential, even fast computers soon get to their limits. Take for example a truth table for a formula with 32 propositional constants. This has $2^{32} = 4294967296$ rows; I'm not sure whether my computer can handle a table of this size efficiently. If you take 265 propositional constants you will get 2^{265} rows, which is roughly the same as the number of atoms in the universe.⁷

Fortunately there are other methods for testing for tautologies, and these methods can also be used for logical systems for which there is no practical method of truth tables. With a lot of additional trickery by computer scientists some proof theories can also be implemented efficiently. Common strands of proof theories are:

- Axiom Systems
- Sequent Calculi
- Natural Deduction
- Semantic Tableaux

All of them are similar to each other and have their advantages and disadvantages depending on what they are used for. For example, axiom systems are often used for capturing informal theories in a formal setting in a first step of formalizing the theory by means of logical tools. Sequent calculi are often used for proving important metatheorems about new logical systems. Natural deduction rules to some extent reflect the way people reason logically (insofar as they do so) and can be regarded as more accessible versions of sequent calculi. Semantic tableaux are easy to use and often provide the basis for the implementation of automatic theorem provers. We will learn how to use semantic tableaux. Once you have mastered semantic tableaux, you will have the skills necessary to understand other proof theories even if you do not appreciate their complexity.

Theorem. A wff that has been proved using proof theoretic methods is called a theorem.⁸

⁷ According to the Wolfram search engine, which gives the number 10^{80} as an estimate.

⁸ Used in a more general sense, the term 'theorem' is also often used for any interesting proposition that is provably true and of a certain importance. Whether a valid formula in some

$\begin{array}{c} \phi \wedge \psi \\ \\ \phi \\ \psi \end{array}$	$\begin{array}{c} \neg(\phi \vee \psi) \\ \\ \neg\phi \\ \neg\psi \end{array}$	$\begin{array}{c} \phi \vee \psi \\ \wedge \\ \phi \quad \psi \end{array}$
$\begin{array}{c} \neg(\phi \wedge \psi) \\ \wedge \\ \neg\phi \quad \neg\psi \end{array}$	$\begin{array}{c} \neg\neg\phi \\ \\ \phi \end{array}$	$\begin{array}{c} \phi \rightarrow \psi \\ \wedge \\ \neg\phi \quad \psi \end{array}$
$\begin{array}{c} \neg(\phi \rightarrow \psi) \\ \\ \phi \\ \neg\psi \end{array}$	$\begin{array}{c} \phi \leftrightarrow \psi \\ \wedge \\ \phi \quad \neg\phi \\ \psi \quad \neg\psi \end{array}$	$\begin{array}{c} \neg(\phi \leftrightarrow \psi) \\ \wedge \\ \phi \quad \neg\phi \\ \neg\psi \quad \psi \end{array}$

Table 2.2: Tableaux rules for propositional logic.

2.3.1 Tableaux Rules for Propositional Logic

A semantic tableaux is an at most binary branching tree containing formulas at each node. Table 2.2 lists the rules for constructing such trees given some initial wff. The tableaux rules are very easy to read. Suppose you have a wff χ with main connective \wedge and conjuncts ϕ and ψ . Each conjunct could be a propositional constant or a complex formula – that is the reason why we’ve used meta variables for elements in our set L_s of well-formed formulas. The rule for \wedge then allows you to put ϕ and ψ under χ within the same branch of the tree as χ . Here is an example of a single application of the rule for conjunction:

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

Nothing branches so far but as you can see the tree will start to branch when we apply the rule for disjunction to $q \vee r$, leading to the following tree:

interpreted logical language or a theorem in general is meant is usually clear from the context, but in case of doubt theorems *about* a logical language should be called *metatheorems*. When the terms are not used for logical languages in particular, a less interesting theorem is usually called *lemma*, *corollary*, or just *proposition*.

$$\begin{array}{c}
 p \wedge (q \vee r) \\
 | \\
 p \\
 q \vee r \\
 \wedge \\
 q \quad r
 \end{array}$$

Let's take a look at another example. We want to prove the wff $\neg(p \wedge \neg p)$. First, we negate it and start a new tree. Then we apply the rules that are applicable. The resulting tree is:

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

First the rule for double negation and then the rule for conjunction was applied. As it happens p and $\neg p$ occur on the same branch. It follows from the rules of our calculus that this means that there is no model for $\neg \neg(p \wedge \neg p)$. From this we can conclude that $\neg(p \wedge \neg p)$ is a theorem. Let's cross-check with a truth table:

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

The main junctor is the outer negation and for all combinations of truth values of the propositional constant p the truth value of the whole formula is 1. Thus, the formula is valid and our result was correct.

2.3.2 How to Use Tableaux

A few more definitions will make our intuitive assessment of how semantic tableaux work more precise.

Complete Tree. A tree constructed with the tableaux rules is complete if the leaves of all branches only contain formulas of the form ϕ or $\neg\phi$, i.e. if no more rules can be applied to formulas to which no rules has been applied so far.

Closed Branch. The branch of a tree is closed if it contains a formula ϕ and its negation $\neg\phi$.

Open Branch. A branch is open if it is not closed.

Closed Tree. A tree constructed with the tableaux rules is closed if all branches of the tree are closed.

Proving a Theorem. In order to prove that a wff ϕ is a theorem, start a new tree with the negated wff $\neg\phi$. Then subsequently extend the tree by applying the rules to all open branches until all branches are closed or the tree is complete. If the tree is closed then ϕ is a theorem; otherwise it is not a theorem.

Provability. When there is a proof for a wff ϕ we write $\vdash \phi$ for the fact that ϕ is provable.

You might wonder why I talk about proving theorems instead of checking for tautologies. I have claimed that the tableaux rules check whether a wff is valid or not, but from a strictly logical point of view this hasn't been shown yet. All I have done is to smear some rules on paper – who knows whether they are the right ones? Proving that they *are* the right ones has two aspects, soundness and completeness, which will be briefly covered in section 2.5. I'll assume these proofs from now on – for propositional calculus and first-order predicate logic they are part of the folklore, whereas you certainly have to show them for any new logical system. If the proof theory is sound and complete it suffices to show $\vdash \phi$ for showing that $\models \phi$ and vice versa.

Tableaux are systematic procedures for finding counter-models, i.e. valuations for all the propositional constants involved for which a given formula is false. A counter-model can also be regarded as the (partial) description of a situation in which the whole formula is definitely false. In order to prove ϕ we then start searching for counter-models for $\neg\phi$, i.e. we begin the tree by assuming $\neg\phi$. When the tree is complete, every branch of the tree that is closed represents a counter-model to $\neg\phi$, i.e. a situation in which $\neg\phi$ is false. If all branches are closed, then $\neg\phi$ is a contradiction. Therefore, ϕ is a tautology. This is a special form of the proof method called *reduction ad absurdum*.

Let us take a look at a sample proof now. We want to prove that $p \rightarrow (p \vee q)$ is a theorem of propositional logic. We prove this by showing that $\neg(p \rightarrow (p \vee q))$ is a contradiction using tableaux:

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

The tree contains only one branch, which is closed by p and $\neg p$. QED.

Let us take a look at an example of a longer proof with a branching tree. Let's prove that $(\neg p \wedge (q \rightarrow p)) \rightarrow \neg q$ is a theorem. This formula exemplifies an argu-

mentation scheme that is often found in real world reasoning.⁹ We assume the negation of this formula and show by using tableaux that it is not satisfiable, i.e. that it is a contradiction:

$$\begin{array}{c}
 \neg((\neg p \wedge (q \rightarrow p)) \rightarrow \neg q) \\
 | \\
 \neg p \wedge (q \rightarrow p) \\
 | \\
 \neg \neg q \\
 | \\
 q \\
 | \\
 \neg p \\
 | \\
 q \rightarrow p \\
 \wedge \\
 \neg q \quad p
 \end{array}$$

Both branches close: The $\neg q$ leaf is in conflict with the q node further above, and the p leaf is in conflict with the $\neg p$ node further above in the tree.

Hints. Remember the following points:

- To prove that a formula ϕ is a tautology, start the tree with $\neg\phi$ and apply the rules until the tree is completed.
- When negating ϕ make sure you negate the whole formula by putting parentheses around it when needed. For example, the negation of $p \rightarrow q$ is $\neg(p \rightarrow q)$.
- You need to evaluate each subformula in each branch unless the branch is already closed.
- If the tree branches into two subtrees and you apply a rule to a formula higher in the tree (i.e. before it branches), you need to put the results into both branches. Here is an example:

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

⁹ Deductive arguments are discussed later. Anyway, here is an example: If it rains, the streets are wet. The streets aren't wet. Hence, it doesn't rain.

- You should always evaluate formulas that do not branch first. The above proof could have simpler been written as:

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

Logical Consequence, Deductive Argument. When ψ can be proved given that the assumptions ϕ_1, \dots, ϕ_n hold, then we write $\phi_1, \dots, \phi_n \vdash \psi$. We say that ψ is a (logical) consequence of ϕ_1, \dots, ϕ_n . When a conclusion ψ logically follows from (is a logical consequence of) a number of premises ϕ_1, \dots, ϕ_n this is also often called a *deductively valid argument*.

In order to proof $\phi_1, \dots, \phi_n \vdash \psi$ using tableaux a new tree is started by putting the premises ϕ_1, \dots, ϕ_n each in a node below each other, add $\neg\psi$, and then continue according to the rules as in case of checking for a tautology. This means that the conjunction of the premises and the negated conclusion is checked for being contradictory. To see that this procedure is sound, consider the simplest case $\phi \wedge \neg\psi$ first. This is equivalent to $\neg(\neg\psi \vee \psi)$ by DeMorgan's law and double negation elimination, which in turn is equivalent to $\neg(\phi \rightarrow \psi)$. If we replace ϕ by the conjunction $(\phi_1 \wedge \dots \wedge \phi_n)$ the same equivalences hold. So the above proof procedure is based on the validity of the scheme $\neg((\phi_1 \wedge \dots \wedge \phi_n) \rightarrow \psi) \leftrightarrow (\phi_1 \wedge \dots \wedge \phi_n \wedge \neg\psi)$.

Deductive Closure. With respect to the consequence relation \vdash of a logical language \mathcal{L} with a set of wffs L we can define the deductive closure $Cn : \mathcal{P}(L) \rightarrow \mathcal{P}(L)$ of a set S of wffs (i.e. of a theory) as follows:

$$Cn(S) := \{\phi \in L \mid \psi_1, \dots, \psi_n \in S \text{ and } \psi_1, \dots, \psi_n \vdash \phi\} \quad (2.1)$$

The deductive closure of a set of formulas represents all formulas that logically follow from that set in a given logical system (such as our language PC), and the concept of deductive closure plays an important role in many applications of logic (e.g. in so-called belief revision theory). A special, deviate case to watch out for is when the set of formulas is contradictory, e.g. $\{p \wedge \neg p\}$. In that case, the result is all sentences of the language, i.e. $Cn(\{p \wedge \neg p\}) = L$, because in classical logic everything follows from a contradiction – and L is, of course, also contradictory as long as the language contains a negation. This issue is discussed in more detail below.

2.3.3 Alternative Notation

Consider this proof from the last section:

$$\begin{array}{c}
 \neg((\neg p \wedge (q \rightarrow p)) \rightarrow \neg q) \\
 | \\
 \neg p \wedge (q \rightarrow p) \\
 | \\
 \neg \neg q \\
 | \\
 q \\
 | \\
 \neg p \\
 | \\
 q \rightarrow p \\
 \wedge \\
 \neg q \quad p
 \end{array}$$

This proof can be represented in a more verbose and failsafe way as follows:

To show: $(\neg p \wedge (q \rightarrow p)) \rightarrow \neg q$

Proof:

1.	$\neg((\neg p \wedge (q \rightarrow p)) \rightarrow \neg q)$	assumption
2.	$\neg p \wedge (q \rightarrow p)$	1: $\neg \rightarrow$
3.	$\neg \neg q$	1: $\neg \rightarrow$
4.	q	3: $\neg \neg$
5.	$\neg p$	2: \wedge
6.	$q \rightarrow p$	2: \wedge
7a.	$\neg q$	7b. p 6: \rightarrow

The tree is closed by 7a, 4 and 7b, 5. QED.

While this notation is more cumbersome, it is less error prone and recommended for larger proofs. Typesetting these tables in \LaTeX is not very pleasant but writing them is not so hard. I often use a large sheet of paper sideways and start with the formula centered in the middle on top.

2.3.4 Selected Theorems

In this section a number of theorems of propositional logic are laid out. Missing proofs are left out for exercise 24.

T1. $\neg(p \vee q) \leftrightarrow (\neg p \wedge \neg q)$

Proof:

1.	$\neg(\neg(p \vee q) \leftrightarrow (\neg p \wedge \neg q))$	assumption
2a.	$\neg(p \vee q)$	2b. $\neg\neg(p \vee q)$ 1: $\neg \leftrightarrow$
3a.	$\neg(\neg p \wedge \neg q)$	3b. $\neg p \wedge \neg q$ 1: $\neg \leftrightarrow$
4a.	$\neg p$	4b. $p \vee q$ 2a: $\neg \wedge$, 2b: $\neg \neg$
5a.	$\neg q$	$\neg p$ 2a: $\neg \wedge$, 3b: \wedge
6a.	$\neg\neg p$	6b. $\neg\neg q$ 6c. $\neg q$ 3a: $\neg \wedge$, 3b: \wedge
7a.	p	7b. q 7c. p 7d. q 6a: $\neg \neg$, 6b: $\neg \neg$, 4b: \vee , 4b: \vee

The tree is closed by (7a, 4a), (7b, 5a), (7c, 5b), and (7d, 6c). QED.

T2. $\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q)$

Proof:

1.	$\neg((p \rightarrow q) \leftrightarrow (\neg p \rightarrow \neg q))$	assumption
2a.	$p \rightarrow q$	2b. $\neg(p \rightarrow q)$ 1: $\neg \leftrightarrow$
3a.	$\neg(\neg q \rightarrow \neg p)$	3b. $\neg q \rightarrow \neg p$ 1: $\neg \leftrightarrow$
4a.	$\neg q$	4b. p 3a: $\neg \rightarrow$, 2b: $\neg \rightarrow$
5a.	$\neg\neg p$	5b. $\neg q$ 3a: $\neg \rightarrow$, 2b: $\neg \rightarrow$
6a.	p	6b. $\neg\neg q$ 6c. $\neg p$ 5a: $\neg \neg$, 3b: \rightarrow , 3b: \rightarrow
7a.	$\neg p$ 7b. q	7c. q 2a: \rightarrow , 2a: \rightarrow , 6b: $\neg \neg$

The tree closes with (7a, 6a), (7b, 4a), (7c, 5b), and (6c, 4b). QED.

T3. $p \rightarrow (p \vee q)$

Proof:

1.	$\neg(p \rightarrow (p \vee q))$	assumption
2.	p	1: $\neg \rightarrow$
3.	$\neg(p \vee q)$	1: $\neg \rightarrow$
4.	$\neg p$	3: $\neg \vee$
5.	$\neg q$	3: $\neg \vee$

The tree is closed by 4, 2. QED.

T4. $((p \wedge q) \vee r) \leftrightarrow ((p \vee r) \wedge (q \vee r))$

Proof: Table 2.3 on page 54. The tree closes with (7a, 4a), (7b, 5a), (6c, 3b), (6d, 3b), (8c, 7c), (8d, 4d), (9a, 7d), (9b, 4d), and (8f, 4d). QED.

T5. $((p \vee q) \wedge r) \leftrightarrow ((p \wedge r) \vee (q \wedge r))$

T6. $(p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$

T7. $p \vee \neg p$

Proof:

0. $\neg(((p \wedge q) \vee r) \leftrightarrow ((p \vee r) \wedge (q \vee r)))$											
1a. $(p \wedge q) \vee r$						1b. $\neg((p \wedge q) \vee r)$					
2a. $\neg((p \vee r) \wedge (q \vee r))$						2b. $(p \vee r) \wedge (q \vee r)$					
3a. $p \wedge q$			3b. r			3c. $\neg(p \wedge q)$					
4a. p			4b. $\neg(p \vee r)$			4c. $\neg(q \vee r)$			4d. $\neg r$		
5a. q			5b. $\neg p$			5c. $\neg q$			5d. $p \vee r$		
6a. $\neg(p \vee r)$			6b. $\neg(q \vee r)$			6c. $\neg r$			6e. $q \vee r$		
7a. $\neg p$			7b. $\neg q$			7c. $\neg p$			7d. $\neg q$		
8a. $\neg r$			8b. $\neg r$			8c. p			8d. r		
									8e. p		
									8f. r		
									8g. q		
									8h. r		
									8i. q		
									8j. r		
									8k. q		
									8l. r		
									8m. q		
									8n. r		
									8o. q		
									8p. r		
									8q. q		
									8r. r		
									8s. q		
									8t. r		
									8u. q		
									8v. r		
									8w. q		
									8x. r		
									8y. q		
									8z. r		

Table 2.3: Tableaux proof of theorem T4.

1. $\neg(p \vee \neg p)$ assumption
2. $\neg p$ 1: $\neg\vee$
3. $\neg\neg p$ 1: $\neg\vee$
4. p 3: $\neg\neg$

The tree is closed by 4,2. QED.

T8. $p \leftrightarrow (p \wedge (q \vee p))$


T9. $p \leftrightarrow (p \vee (q \wedge p))$

T10. $((p \wedge q) \rightarrow r) \leftrightarrow (p \rightarrow (q \rightarrow r))$

T11. $((p \rightarrow q) \rightarrow p) \rightarrow p$

T12. $((p \rightarrow q) \wedge (\neg p \rightarrow r)) \leftrightarrow ((p \wedge q) \vee (\neg p \wedge r))$

2.3.5 Exercises

 **Exercise 24** Prove the following theorems using semantic tableaux:

- | | |
|-----------|------------------|
| a. T5, T6 | c. T10, T11, T12 |
| b. T8, T9 | |

2.4 Deductive Arguments

2.4.1 Valid Argument Schemes

Deductively Valid Argument Schemes. When it has been shown that a particular wff is a logical consequence of a number of premises, this holds in general. For example, since $p, p \rightarrow q \vdash q$ is provable $q, q \rightarrow p \vdash p$ holds as well. This can be shown by conducting the proof using meta-variables for propositional constants such as ϕ and ψ . We can therefore speak of deductively valid *argument schemes*.

In case you are in doubt about the validity of some purported argument scheme use truth tables or tableaux for checking whether the scheme is valid. As opposed to what was usual in the Middle Age there is no longer any need for learning dozens of arcane names for valid or invalid argument schemes by heart. Nevertheless, the following names for valid argument schemes are so common that you should know them:

- Modus Ponens: $\phi, \phi \rightarrow \psi \vdash \psi$

- Modus Tollens: $\neg\psi, \phi \rightarrow \psi \vdash \neg\phi$
- Contraposition: $\phi \rightarrow \psi \vdash \neg\psi \rightarrow \neg\phi$

2.4.2 Sound Arguments, Fallacies, Good Arguments

Of course, in order to argue correctly – not to speak of convincingly, which is an entirely different matter! – it does not suffice to use a deductively valid argument scheme. The arguer also has to make it plausible to the audience that the premises are true. Although it might have happened occasionally in the humanities (e.g. in philosophy), correctly deducing in painstaking detail a conclusion from completely idiotic premises is quite pointless.

Sound Argument. A *sound argument* is a deductively valid argument whose premises are true.

Since in many cases truth cannot be established with absolute certainty this definition must be weakened to be useful in real life. An arguer must attempt to make the premises as plausible as possible such that the audience agrees with them. Once this has been achieved real-world arguments that are also deductively valid are usually considered sound. Of course, people often disagree about the truth of the premises and so soundness of an argument is generally harder to establish as deductive validity. Consider for example the following deductive argument:

- God is the most supreme being.
- If something is the most supreme being, then it must also have the property of existing.
- Therefore, God exists.

In propositional logic this argument can only be approximated. Let p be read as *God is the most supreme being* and q be read as *God exists*. The argument is then just an application of modus ponens $p, p \rightarrow q \vdash q$ and clearly deductively valid. But perhaps not everyone buys into the premises.

On a side note, not only the quantification has been ignored by the translation to propositional logic. Suppose the gradable adjective ‘supreme’ expresses a preorder relation. If this is so, one might argue that there may be more than one most supreme being. When using the superlative of a gradable adjective people commonly assume that there is only one object satisfying the superlative, yet this need not be so. It certainly doesn’t follow from the appealing assumption that the meaning of the adjective is based on a preorder. Consider for example ‘tallest mountain’. Although it is natural to speak of ‘*the* tallest mountain’ there could be two or more mountains with exactly the same height.

Mathematically speaking, a preorder on a set does not in general ensure the existence of a single minimum or maximum element in the set with respect to that order. The set could be infinitely extending in one or both directions (called an *infinite chain*) or there could be several elements in the maximum or minimum that are equivalent with respect to the equivalence relation generated by the preorder. Right? So the above argument is incomplete in a nontrivial way. I mention this example in order to make clear that there is much more about deductive arguments than just their deductive validity and that there always remains room for disputes about the soundness of the premises and the correct interpretation of the argument.

Logical Fallacy. In argumentation sometimes deductive argument schemes are applied that are not valid. If someone uses such an invalid argument, he commits a *logical fallacy*.

Here is a typical logical fallacy:

	If it rains the street is wet.
	It doesn't rain.

☹	Therefore, the street is not wet.

This popular fallacy is sometimes called *denying the antecedent*. The fallacy is of the form $((\phi \rightarrow \psi) \wedge \neg\phi) \rightarrow \neg\psi$. You know enough about logic to see that this scheme is not valid, but in case of doubt you may check it using a truth table or semantic tableaux. Here is another common logical fallacy:

	If it rains the street is wet.
	The street is wet.

☹	Therefore, it rains.

This fallacy is called *affirming the consequent*. The inference is not valid, as there could be a multitude of other reasons why the street is wet. The fallacy is of the form $((\phi \rightarrow \psi) \wedge \psi) \rightarrow \phi$. Again, you can check the invalidity of this argument scheme easily using truth tables or tableaux. It is, however, important to bear in mind that even deductively invalid argument schemes may convey useful information about the subject matter or may confirm the conclusion to some degree. For example, in the absence of any reasonable alternative antecedents ϕ_i , where $\phi_i \rightarrow \psi$, assuming that ψ suggests ϕ can make sense. (See note 9 below.) Likewise, even if $(p \wedge q) \rightarrow r$ and p does not allow you to conclude r , the two formulas tell you that you might want to check whether q holds in order to establish r .

✧ **Remark 5 (Good Arguments)** There is much more to putting forward a good argument than just using a deductively valid argument scheme and making the premises plausible. A good presentation is just as important. On one hand, a logical fallacy will never make a good deductive argument. On the other hand, it is not reasonable to demand that *any* good argument must be deductively valid. It is quite common in real argumentation to leave premises implicit and making all premises explicit would sometimes not be economical and make the argument long and dull. Moreover, truth-conducive albeit fallible heuristic reasoning patterns are used both in science and everyday reasoning with some success and discarding them altogether would be too strict. See Note 9 for more information.

Deductively valid arguments are *truth-preserving*: If the premises are true then the conclusion is also true. Deductively valid arguments are also *monotonic*, because classical logical consequence is monotonic: If $\phi_1, \dots, \phi_n \vdash \psi$ then $\phi_1, \dots, \phi_n, \phi_{n+1} \vdash \psi$. Notice that there are two cases in such a situation:

- The additional premise ϕ_{n+1} is consistent with ϕ_1, \dots, ϕ_n , i.e. $\phi_1 \wedge \dots \wedge \phi_n \wedge \phi_{n+1}$ is satisfiable. Then ϕ_{n+1} is redundant. (But it is also possible that one of ϕ_1, \dots, ϕ_n is redundant when we decide to keep ϕ_{n+1} .)
- The additional premise ϕ_{n+1} is inconsistent with ϕ_1, \dots, ϕ_n , i.e. $\phi_1 \wedge \dots \wedge \phi_n \wedge \phi_{n+1}$ is a contradiction. Then ψ follows because anything follows from a contradiction, but the argument is not sound.

The rule of classical logic that anything follows from a contradiction, i.e. $\phi \wedge \neg\phi \vdash \psi$, is sometimes called *ex falso quod libet*. It has given rise to numerous alternative accounts of logical consequence and alternative logical systems. Notice that there is nothing mysterious about the fact that $(\phi \wedge \neg\phi) \rightarrow \psi$, although people have sometimes confused the above problem as one about the meaning of the conditional and have given it the misleading label ‘paradox of the material implication’. The truth-function \rightarrow , however, is simply defined the way it is and is one of the 16 possible binary connectives of classical bivalent logic. Messing around with \rightarrow is therefore bound to get you into troubles at one point or another, usually due to the interdefinability of truth-functions or the lack thereof when \rightarrow has been tweaked in a non-classical logic.

Regarding logical consequence, however, *ex falso quod libet* does indeed not seem to be a very desirable rule. Still it seems wise not to change the notion of logical consequence itself but rather emphasize that there is much more to

a good argument than just being deductively valid.¹⁰ Since *ex falso quod libet* never leads to a sound argument, the problem is not very pressing. Likewise, valid argument schemes like $\phi \vdash \phi \vee \psi$ have been overly criticized. From a logical point of view they do no harm, although they may lead astray and violate principles of sound argumentation.

Notice, finally, that establishing the soundness of premises is not monotonic: You might have good reasons for believing premises ϕ_1, \dots, ϕ_n at some time but when you take into account additional evidence this might weaken some of the premises you've previously considered well-established. The relation between evidence, premises, and possible degrees of plausibility or credibility attributed to them cannot be modeled adequately in classical bivalent logic.

☞ **Note 9 (Induction and Abduction)** Classical deductive arguments represent a strictly monotonic and truth-preserving reasoning pattern. No matter what additional information is taken into account, as long as the premises of a valid deductive argument are true the conclusion will hold. This reasoning plays a crucial role both in 'everyday' argumentation and particularly in mathematical reasoning. There are two other reasoning patterns that are sometimes being regarded as being indispensable to gather knowledge: *abduction* and *induction*.

Inductive arguments form the basis of genuine probabilistic reasoning. From a small number of observed occurrences of events it is inferred that the observed frequencies hold for the respective classes of events in general. This kind of reasoning is non-monotonic, because the inference might no longer hold after additional observations have been made. While it had been controversial for a long time whether inductive reasoning is acceptable in general or whether it is really needed, this issue has been pretty much settled with the advent and success of sophisticated statistical methods and developments in physics according to which genuine randomness can be found in nature.

On the other hand, abduction is still much more controversial. Abduction is also sometimes also called *inference to the best explanation* and has for a long time been discussed under the general idea of a 'logic of discovery'. One strand of abduction, the most popular one, is indirectly based on the fallacy of affirming the consequent. Assume there are hypotheses p_1, p_2, \dots, p_n such that $p_i \vdash c$ and c is observed to be the case, whereas it is not known

¹⁰ If you change the interpretation of \vdash in the meta language (and correspondingly, the proof theory) but keep the truth-functions in the object language classical, the deduction theorem might no longer hold, and that might not be desirable. This theorem says that if $A \cup \{\phi\} \vdash \psi$ then $A \vdash \phi \rightarrow \psi$ for a set of wffs A , i.e. if you can prove the succedent of a conditional from a set of assumptions plus the antecedent of the conditional, then you can prove the whole conditional from the assumptions alone.


which of the hypotheses holds. By an abductive inference one concludes from c that the *most plausible* p_i , say p_3 , is the case. Unless only finitely many premises can and need to be taken into account, which rarely happens, abduction is also non-monotonic, as there could be some premise p_m ($m > n$) that is more plausible than p_3 . Formally, abduction can be implemented by defining ordering the set of possible hypotheses with a preorder relation. The crucial problem is, however, where this ordering comes from, i.e. what ‘most plausible’ means in this context. Note that abduction seems to be pretty common in everyday reasoning, though: Lady Buttersworth has been murdered with an axe. Lord Worcester was found in the same building with a bloody axe in his hand, and the DNA of the blood on the axe matches that of Lady Buttersworth. Hence, Lord Worcester murdered Lady Buttersworth – unless the dear old Lord found the poor Lady, grabbed an axe from the wall with the intention of pursuing the fleeing would-be assassin and thereby contaminated the axe with her blood. Or, perhaps someone planted false evidence on him. (‘My dear, would you mind holding this bloody axe for me for a minute or two, please? I’ll be *right* back...’ – ‘Of course not, Sir Farnsdale. I shall guard this marvelous gardening instrument until you return.’)

2.4.3 Exercises

✎ **Exercise 25** Translate the following deductive argumentation fragments into propositional logic, idealizing the statements as is deemed appropriate, and determine whether they are deductively valid or based on a fallacy.

- a. If John did it, so did I. I didn’t do it, so John didn’t do it either.
- b. Either Jack the Ripper or someone imitating him was responsible for this crime. If Jack the Ripper had done it, the cut would have been deep and precise. The cut is not deep and precise, hence the crime was committed by someone imitating Jack the Ripper.
- c. Either Bob ate all the cookies or it is not the case that Alice went swimming. There are still some cookies left, so Alice went swimming.
- d. John is both a chess player and an excellent boxer. If someone is a chess player, then he is not physically strong. Therefore, John is not physically strong.
- e. Something that nobody likes is bad. Therefore, Capitalism is good. For many people like it and if it were bad, nobody would like it.

- f. If Sir John was killed by a blunt instrument like a Vase or a candlestick, then he died of a head trauma. He did not die of a head trauma. Therefore, Sir John was not killed by a blunt instrument.
- g. John likes Deep Purple or Led Zeppelin. If he likes Deep Purple, he is stupid. He is not stupid, so he likes Led Zeppelin.
- h. That English proper names are rigid is a contingent fact of the English language and not a consequence of the Millian view. For suppose it was a consequence of the Millian view; then any formal or informal implementation of the Millian view in which proper names or their formal counterparts were non-rigid would be inconsistent, but this is clearly not the case.

 **Exercise 26** CIA special analyst Jack Thompson is running queries on a big secret geospatial database in order to track down the whereabouts of known terrorists. The search engine works with boolean queries and can be used to confirm whether a given conclusion follows from a number of geospatial conditions expressed in propositional logic. After simplifying the queries for ‘Osama bin Laden’ and eliminating a number of possibilities, he obtains the following test condition:

```
NOT (NOT In(Osama, province:Nimruz) OR NOT In(Osama,
province:Balochistan, loc:Bolan Pass) OR (In(Osama,
province:Nimruz) OR NOT In(Osama, province:Balochistan,
loc:Bolan Pass)))
```

Jack Thompson enters his query ‘?In(Osama, province:Balochistan, loc: Bolan Pass)’ and obtains the answer ‘YES’. Full of excitement he presents the result to his superiors, who immediately request an airstrike. The airstrike is conducted swiftly, all travelers at the Bolan Pass, Balochistan, are killed and the pass is reduced to rubble. A later investigation concludes that the operation was a failure and among the many civilian victims no traces of Osama bin Laden’s DNA were found.

- a. Show what mistake Jack Thompson made when he queried the database.
- b. During the investigations representatives of the company that created the database and inference engine claim that their software functioned correctly and Jack Thompson misused it. Can you suggest a way how Jack Thompson could have prevented the disaster by entering another query and without changes to the costly geospatial database?
- c. Is the claim of the company justified or can you suggest a technical change to the inference engine that, notwithstanding any other bugs it might have, would prevent such scenarios from occurring in the future?

✎ **Exercise 27** (difficult) Implement the syntax and denotational semantics for an abductive conditional operator $M \models \phi \rightsquigarrow \psi$ that is true if ψ is among the ‘most plausible’ wffs such that $M \models \psi \rightarrow \phi$ holds.

2.5 Metatheorems

Proofs of important metatheorems go beyond the scope of this introduction. However, even for just applying a logical system as a theoretical tool it is necessary to be aware of its most important properties. For propositional logic with a suitable proof theory like the tableaux rules of the previous section it is well-known that the following metatheorems hold. In what follows, the term ‘logical system’ is meant as the ensemble of a logical language, its denotational semantics, and its proof theory.

Soundness. A logical system is *sound* iff. it follows from $\vdash \phi$ that $\models \phi$, i.e. if ϕ can be proved then ϕ is also true in all models.

Completeness. A logical system is *complete* iff. from $\models \phi$ it follows that $\vdash \phi$, i.e. if ϕ is true in all models then ϕ can also be proved.

Decidability. A logical system is *decidable* iff. there is an algorithm to decide in finite time of a wff whether or not it is a tautology.

Compactness. Let a set Γ of wffs be satisfiable if there is a model M such that $M \models \phi_i$ for all $\phi_i \in \Gamma$. A logical system is *compact* iff. any set Γ of wffs of that system is satisfiable if and only if every finite subset $A \subset \Gamma$ is satisfiable.

Propositional logic with the tableaux rules of section 2.3.2 is sound, complete, decidable, and compact.

2.6 Concluding Remarks

Propositional logic might seem overly simplified in order to be of much use but in fact it has many useful applications. For one thing, it is fair to say that calculations with numbers in computers work on the basis of propositional logic (see Note 6).

There is also a rich tradition of using propositional logic in the study of argumentation that has started with Aristotle’s writings. At his time propositional logic and a limited form of quantification in the form of syllogisms were already known and since then have been studied extensively. The most important thing about propositional logic is, however, that it is the basis for other important logical systems like modal logic and first-order predicate logic. The latter is the topic of the next chapter.

✧ **Remark 6 (Computers and Propositional Logic)** Almost all computers represent numbers in the binary system, using only 1 and 0. So the number 0 is represented by 0, the number 1 by 1, the number 2 by the sequence $\langle 1, 0 \rangle$, the number 3 by the sequence $\langle 1, 1 \rangle$, the number 4 by the sequence $\langle 1, 0, 0 \rangle$, the number 5 by the sequence $\langle 1, 0, 1 \rangle$, and so on.¹¹ You get the picture. Now let us represent these sequences by propositional constants p_0 for the least significant bit representing 0 and 1 and p_1 representing 2 if it is true, 0 otherwise, p_2 representing 4 if it is true and 0 otherwise, and so on. According to this convention for example the number 5 can be represented by a model of propositional logic in which the formula $(p_2 \wedge \neg p_1) \wedge p_0$ is true. Consider now arithmetic operations like, for example, addition. Addition of binary numbers without overflow can be represented by $\dot{\vee}$: $0 + 0 = 0$, $0 + 1 = 1$, and $1 + 1 = 0$. See table 2.4 for an example. When an overflow occurs $1 + 1 = 0$ an overflow bit, say c_n , has to be stored that is taken into account when adding the next significant bit. So for example $p_0 \dot{\vee} q_0$ represents the addition of the first bit of p and q without overflow and the overflow is $c_0 := p_0 \wedge q_0$. For the next bit $(p_1 \dot{\vee} q_1) \vee c_0$ represents the result of the addition without overflow and the overflow is $c_1 := ((p_1 \wedge q_1) \vee (p_1 \wedge c_0)) \vee (q_1 \wedge c_0)$ – and so on for the other bits. The resulting representations of arithmetic operations can often be simplified extensively and are implemented in hardware and combined into more complex components. Figure 2.1 shows what is called an ‘adder’. Other operations of computers are implemented in a similar way.

However, you should bear in mind that the example is simplified in many ways. Notice that input and output as well as how storage works has been ignored. In order to describe the workings of computers in general a more expressive logical system than propositional logic is needed, such as, for example, so-called *untyped λ -calculus*. To avoid potential misunderstandings, the version of λ -calculus to be introduced in chapter 4 is typed and not expressive enough for that purpose. To put it in technical terms, simple untyped λ -calculus is Turing complete whereas simple typed λ -calculus is not.

¹¹ Obviously, it is merely a matter of convention/implementation whether the least significant bit is the leftmost or rightmost bit of such a sequence. And in fact both orders of representation have been used in different types of microprocessors, and failures to convert numbers between the two format when they were transferred from one system to another has occasionally caused problems. As you probably know, the issue is further complicated by the fact that modern computers don’t store numbers as sequences of arbitrary length but always in blocks of 4, 8, 16, 32, 64 or 128 bits.

(a)	0	0	1	0
	0	1	0	1
<hr/>				
	0	1	1	1

(b)	0	0	1	1
	0	1	0	1
<hr/>				
	1	1	1	1
	1	0	0	0

Table 2.4: Binary representation of (a) $2 + 5$ and (b) $3 + 5$. The rightmost bit is the least significant. So for example, $5 = (1 \cdot 1) + (0 \cdot 2) + (1 \cdot 4) + (0 \cdot 8)$.

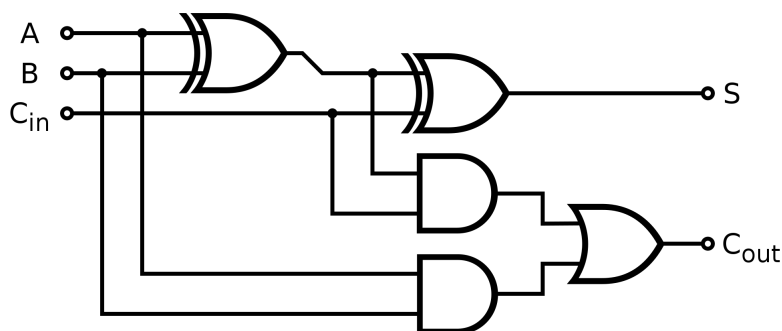


Figure 2.1: Circuit diagram of an adder. (Source: Wikimedia Commons)

2.7 Literature

Just about any introduction to logic covers propositional logic and regarding the basics introductions to logic only differ in perspective and in the level of formal rigidity. Here are some classics that are less or at most as formal as this introduction:

- Alfred Tarski (1995). *Introduction to Logic*. Dover.
- Wilfried Hodges (1977, 2001). *Logic: an introduction to elementary logic*. Penguin.

I highly recommend the book by Hodges, which is easy to read and covers a vast range of material that is particularly relevant for linguists. He also uses tableaux as a proof theory. Other seminal texts can be found in the literature section of the next chapter. Please notice that books on ‘informal logic’ or ‘critical thinking’ are *not* suitable for learning logic.

First-Order Logic

This chapter is about first-order predicate logic (often abbreviated *FOL*). In addition to predicate logic it allows for predicates, relations, constants and variables for individuals and quantifiers like \forall with reading *for all* and \exists with reading *there is at least one*. Like in the last chapter, we first introduce the syntax of the formal language and then its model-theoretic semantics. Finally, tableaux rules will be introduced as a proof theory.

3.1 Syntax of First-order Predicate Logic with Identity

Let us call our version of first order predicate logic *FOL*. Usual formulations contain constants for predicates, constants and variables for a domain, and the quantifiers in addition to the connectives of propositional logic. Having functions in the object language is optional, as they can be defined as restricted relations. Many authors don't include functions, as they make definitions a bit more complicated. (See for example the definition of terms below.)

3.1.1 Basic Expressions.

Property, Relation, and Function Symbols. Let L_R be the set of property and relation symbols consisting of sequences of letters starting with a capital letter, usually just P, Q, R, S and their indexed variants.¹ All symbols in L_R have a fixed *arity*, which is an integer number n ($1 \leq n$). We stipulate that $=$ of arity 2 is in L_R . Let L_F be the set of function symbols consisting of f, g, h and

¹ We will also sometimes use predicate symbols like *Man* or *Mortal* in order to indicate their intended interpretation.

their indexed variants. Function symbols also have an arity, which is a positive integer number.

Terms. Let L_V be the set of variables consisting of x, y, z and their indexed variants. Let L_C be the set of individual constants consisting of a, b, c, d and their indexed variants. Let L_T be the set of terms defined as follows. If $\alpha \in (L_C \cup L_V)$ then $\alpha \in L_T$. If $\alpha_1, \dots, \alpha_n \in L_T$ and $\sigma \in L_F$ and of arity n , then $\sigma(\alpha_1, \dots, \alpha_n) \in L_T$.

Ground Term. A term that does not contain any variable is a *ground term*. For example, $a, b, f(a), g(f(a, b), c)$, and $g(g(a))$ are ground terms and $x, y, f(a, x), g(f(a), y)$, and $g(g(x))$ are not ground terms.

Connectives. The set of truth-functional connectives is the same as for propositional logic, i.e. $L_B := \{\vee, \wedge, \rightarrow, \leftrightarrow, \uparrow, \downarrow, \dot{\vee}\}$ and $L_N := \{\neg\}$.

Quantifiers. The set of quantifiers is defined as $L_Q := \{\forall, \exists\}$.

3.1.2 Well-formed Formula.

The set L_S of well-formed formulas of *FOL* is defined as follows:

1. If $P \in L_R$ and of arity n ($n > 0$) and $\alpha_1, \dots, \alpha_n \in L_T$ then $P(\alpha_1, \dots, \alpha_n) \in L_S$.
Example: $R(a, x, b)$ (arity 3)
2. If $\phi \in L_S$ and $v \in L_N$, then $v\phi \in L_S$.
Example: $\neg P(x, y)$
3. If $\phi, \psi \in L_S$ and $\circ \in L_B$ then $(\phi \circ \psi) \in L_S$.
Example: $(P(a) \wedge R(x, a))$
4. If $\phi \in L_S$, $x \in L_V$, and $\xi \in L_Q$ then $\xi x(\phi) \in L_S$.
Example: $\exists x(P(x) \vee Q(x, y))$
5. Nothing else is in L_S .

This time I have sacrificed correctness a little bit for better readability and used P both as a meta-variable for relations and a symbol for a relation or predicate and x both as a meta-variable for variables and the name of a variable. According to the above rules for example the following formulas are wff: $P(x)$, $P(a) \wedge P(a, b)$, $\exists x P(a)$, $\forall y (\exists x (R(f(x), y) \vee R(y, g(x, y))))$, $\exists x (P(x) \wedge \forall y Q(y, a))$.

✧ **Remark 7 (Simpler definition of the syntax)** The above definition of the syntax is a bit complicated. The idea is that you should be able to correctly formulate a syntax on the basis of sets for syntactic entities, as this can be useful for certain kinds of proofs, and are also also understand similar definitions in the literature. With a bit less precision and using object-language expressions as metavariables the same syntax as the above one could also be defined as follows:

- Terms
 - x, y, z and indexed variants are variables, P, Q, R and indexed variants are predicates with a given arity, a, b, c, d, e and indexed variants are constants, and f, g, h and indexed variants are function symbols with a given arity.
 - Variables and constants are terms; if f is a function symbol of arity n and t_1, \dots, t_n are terms, then $f(t_1, \dots, t_n)$ is also a term.
- Well-formed Formulas
 - If P is a predicate of arity n and t_1, \dots, t_n are terms, then $P(t_1, \dots, t_n)$ is a wff.
 - If A is a wff then $\neg A$ is also a wff.
 - If A and B are wffs then $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \leftrightarrow B)$, $(A \dot{\vee} B)$, $(A \dot{\wedge} B)$, and $(A \dot{\uparrow} B)$ are also wffs.
 - If A is a wff and x is a variable, then $\exists x(A)$ and $\forall x(A)$ are also wffs.

Notational Conventions. Redundant parentheses may be left out. Square brackets [and] may be used instead parentheses (and). The identity sign = may be used in infix notation, i.e. we generally write $a = b$ instead of $=(a, b)$, and we write $x \neq y$ for $\neg = (x, y)$. A dot . opens a parenthesis that is closed as rightmost as possible in the formula. Without dot or enclosing parentheses the argument of a quantifier is always taken as the shortest wff, i.e. for example $\exists x P x \wedge Q x$ must be read $\exists x [P x] \wedge Q x$. The arguments of a predicate or relation may be written without enclosing parentheses. Conjunction and disjunction bind stronger than conditional and biconditional. Several subsequent applications of \exists or \forall may be contracted.

✧ **Example 6 (Shortcut Notation)**

1. $P x \wedge Q x \rightarrow R x$ may be written for $((P(x) \wedge Q(x)) \rightarrow R(x))$

2. $\exists x.P(x, y)$ may be written for $\exists x(P(x, y))$
3. $Px, y \vee Py, x$ may be written instead of $P(x, y) \vee P(y, x)$
4. $\forall x \exists y.Px \rightarrow R(x, y) \wedge Qx$ may be written for $\forall x \exists y((P(x) \rightarrow (R(x, y) \wedge Q(x))))$
5. $\forall xyz.P(x, y, z)$ may be written for $\forall x \forall y \forall z P(x, y, z)$

There is no need to use the shortcut notation and if you feel uncomfortable with it the best thing to do is to resort to full bracketing and add missing parentheses in formulas. Both too many and too few parentheses can confuse at times, and the right balance is a matter of personal taste. Notice that some logicians or logic teachers do not like or allow certain shortcut notations. In particular, for whatever reason some people seem to wholeheartedly dislike writing Px, y instead of $P(x, y)$ and I will only use this notation sparingly.

☞ **Note 10 (Yet another way to specify the syntax)** Using a very dense notation the above grammar could have been specified as

$$S := R_n(x_1, \dots, x_n) \mid (S \wedge S) \mid \neg S \mid \forall x S,$$

where the remaining functors and quantifiers would have to be defined by abbreviation.

Free vs. Bound Variables. A central concept in first-order logic is the distinction between free and bound variables. This distinction is purely syntactical but has important semantic ramifications. It is best to define free and bound variables recursively based on the structure of wffs. Here is how. Let $\text{fvar} : (L_S \cup L_T) \rightarrow \mathcal{P}(L_V)$ be a function determining the free variables of a wff and $\text{bvar} : (L_S \cup L_T) \rightarrow \mathcal{P}(L_V)$ a function determining the bound variables of a wff as follows:

1. If $\alpha \in L_V$ then $\text{fvar}(\alpha) = \{\alpha\}$ and $\text{bvar}(\alpha) = \emptyset$.
2. If $\alpha \in L_C$ then $\text{fvar}(\alpha) = \emptyset$ and $\text{bvar}(\alpha) = \emptyset$.
3. If $P \in L_R$ and of arity n ($n > 0$) and $\alpha_1, \dots, \alpha_n \in L_T$ then $\text{fvar}(P(\alpha_1, \dots, \alpha_n)) = \text{fvar}(\alpha_1) \cup \dots \cup \text{fvar}(\alpha_n)$ and $\text{bvar}(P(\alpha_1, \dots, \alpha_n)) = \emptyset$.
4. If $\phi \in L_S$ and $v \in L_N$, then $\text{fvar}(v\phi) = \text{fvar}(\phi)$ and $\text{bvar}(v\phi) = \text{bvar}(\phi)$.
5. If $\phi, \psi \in L_S$ and $\circ \in L_B$ then $\text{fvar}(\phi \circ \psi) = \text{fvar}(\phi) \cup \text{fvar}(\psi)$ and $\text{bvar}(\phi \circ \psi) = \text{bvar}(\phi) \cup \text{bvar}(\psi)$.

6. If $\phi \in L_S$, $x \in L_V$, and $\xi \in L_Q$ then $\text{fvar}(\xi x(\phi)) = \text{fvar}(\phi) \setminus \{x\}$ and $\text{bvar}(\xi x(\phi)) =$
- $$\begin{cases} \text{bvar}(\phi) \cup \{x\} & \text{if } x \in \text{fvar}(\phi) \\ \text{bvar}(\phi) & \text{otherwise} \end{cases} .$$

As scary as this definition might seem at first glance, it is only a precise and mathematically correct way of defining what is intuitively easy to see. Intuitively, in a formula like $\exists x.P(x, y)$ the variable x is bound by the existential quantifier \exists , whereas y remains free. If we only look at $P(x, y)$, on the other hand, intuitively both variables are free (=not bound). Now how does the above definition work? Clause 1 puts every variable into the set of free variables. Constants contain neither bound nor free variables, which is stated by clauses 2. Clauses 3 to 5 then just accumulate the sets of free and bound variables for compound expressions by taking into account the free and bound variables of their parts. The final and crucial clause 6 says that a quantifier binds a variable; so if the variable was previously free it is now no longer free, i.e. removed from the set of free variables, and put in the set of bound variables. To understand the case distinction in this definition, consider the following formulas, which are wffs but ‘strange’:

- $\forall x.Pa$
- $\forall x.Px \rightarrow \exists x[Qx]$
- $\forall x\exists x.Px$

The first wff illustrates *vacuous quantification*. It has no semantic effect but we haven’t disallowed it in our syntax. The second wff illustrates a case of *variable reuse*. This is usually allowed and some authors love to use it extensively in order to confuse their readers. The last wff illustrates both vacuous quantification and variable reuse. The universal quantifier doesn’t have any effect, because the variable is immediately rebound by the existential quantifier. The case distinction of the above definition formally captures the idea that in case of vacuous quantification the variable is not really bound. A wff may contain unbound variables, in case of which it is open:

Open Formula. A formula A that contains variables that are free in A is called an *open formula*.

Finally, a further important syntactic concept is that of the scope of a quantifier, which is indicated by the parentheses around the quantifier body. I only specify it informally as follows.

Scope. A variable is in the scope of the occurrence of a quantifier if it could be bound by the occurrence of that quantifier.

3.1.3 Exercises

✎ **Exercise 28** Determine which of the following terms are ground:

- | | | |
|--------------------|------------------------|--------------------|
| a. $f(a, b, x)$ | d. $f(f(f(a)))$ | g. $f(g(a), h(a))$ |
| b. c | e. $g(c, f(a))$ | |
| c. $f(g(x, a), b)$ | f. $h(x_{90}, y_{29})$ | |

✎ **Exercise 29** Indicate the scope of the quantifiers in the following formulas by underlining and overlining them as in the example. (You may also use colors, of course.)

Example: $\forall x.P(x) \wedge \exists y[\overline{P(y) \vee Q(x, y)}]$

- | | |
|--|--|
| a. $\forall xPx \rightarrow \exists xPx$ | g. $\exists xPx \wedge \forall y.Py \rightarrow x = y$ |
| b. $\forall x.Px \rightarrow \exists xPx$ | h. $P(x, y) \wedge \forall x\forall y[P(x, y)]$ |
| c. $\forall x.Px \rightarrow \exists yPy$ | i. $\forall xyP(x, y) \wedge R(x, y)$ |
| d. $\forall x.Px \rightarrow \exists yQ(x, y)$ | j. $\forall x\exists y.Px \rightarrow P(f(x, y))$ |
| e. $\forall xyz.Px \wedge [R(x, z) \rightarrow Q(y, z)]$ | k. $\neg\forall x\exists y[x = f(y)]$ |
| f. $\forall xyz[R(x, y) \wedge R(y, z) \rightarrow R(x, z)]$ | |

3.2 Semantics of First-Order Logic with Identity

We will now specify the semantics of first-order logic. Not very surprisingly, the connectives are defined in exactly the same way as in propositional logic. However, we need rules additional for predication and for at least one quantifier. We have in a sense already learned how to interpret first-order logic in the first chapter and only have to apply this knowledge now.

3.2.1 Variable Assignments and Variants

Assignment. An variable assignment is a function $g : L_V \rightarrow D_e$, where D_e is a set of individuals specified by a model (see below).²

Variant of an Assignment. An x -variant h of an assignment g is the same function as g except that it is possible that $h(x) \neq g(x)$. As a shortcut, we write $h \approx_x g$ for the x -variant h of g .

² I use e for ‘entities’ according to a convention from higher-order logic. More on that in the next chapter.

Notice that $h(x) = g(x)$ is not disallowed when $h \approx_x g$; it is only *possible* that $h(x) \neq g(x)$.

3.2.2 Models and Truth in a Model

Model. A model for *FOL* is an ordered pair consisting of a non-empty domain D_e for individuals and an interpretation function $I^g(\cdot)$ that maps non-logical constants to their extension in dependence of a variable assignment g as follows:

1. Variables: If $\alpha \in L_V$ then $I^g(\alpha) = g(\alpha)$ where $g(\alpha) \in D_e$.
2. Constants: If $\alpha \in L_C$ then $I^g(\alpha) \in D_e$.
3. Functions: If $\xi \in L_F$ and of arity n then $I^g(\xi)$ is an n -ary total function $f : D_e^n \rightarrow D_e$, i.e. a function taking an n -tuple $\langle a_1, \dots, a_n \rangle$ ($a_1, \dots, a_n \in D_e$) and yielding a $b \in D_e$.
4. Properties, Relations: If $P \in L_R$ and of arity n then $I^g(P) \subseteq D_e^n$, i.e. a subset of the set of n -tuples over D_e .

N.B. It is common to leave out g in cases when it is not relevant. (The assignment is only used for variables.) Also, sometimes authors use separate functions for term interpretations and the interpretation of relations, but this distinction is usually not made explicit in the model. It is very common not to include functions, as they are not essential to the logic.³

Watch out the requirement of the above definition that the domain D_e is non-empty, which of course means that $D_e \neq \emptyset$. As innocuous as it might seem, this requirement is important and omitting it would have unexpected consequences.

◇ **Remark 8 (Propositional Constants in First-order Logic)**

Sometimes authors allow relation symbols of arity 0, which are effectively interpreted as propositional constants. We do not do this here, because it unnecessarily complicates the syntax and semantics. However, as long as tense is ignored it may be argued that English uses of the *expletive* ‘it’ sometimes take no arguments. Here is a typical example:

(3.1) It rains.

The pronoun ‘it’ in this example may be regarded as a pseudo-subject in the sense that it is grammatically in subject position but does not indicate the presence of a *logical subject* to which the property of raining would be ascribed. In Portuguese this is clearer, since an expletive use of ‘ele’ is not grammatical:

³ The functions are total and we have seen in the first chapter that they can be defined in terms of a suitably restricted relation.

(3.2) Chove.

(3.3) * Ele chove.

When tenses or situations are taken into account the predicate for ‘to rain’ and ‘chover’ will likely take other arguments, though. For example, in event semantics the above examples could be expressed as $\exists e.R(e)$, where e is a special sort of variable that stands for an event. So in such a framework the paraphrase for ‘chove’ would be *there is a raining event*.

Truth in a Model. The evaluation function \models maps wffs in L_S to $\{1, 0\}$ in a model M in dependence of an assignment g . To avoid misunderstandings I specify the complete definitions this time, including the clause for 0 (false) that is often omitted. This time we use $\llbracket \cdot \rrbracket$ as a symbol for evaluation instead of \models .⁴

$$\llbracket P(\alpha_1, \dots, \alpha_n) \rrbracket^{M,g} = \begin{cases} 1 & \text{if } \langle I^g(\alpha_1), \dots, I^g(\alpha_n) \rangle \in I^g(P) \\ 0 & \text{otherwise} \end{cases} \quad (3.4)$$

$$\llbracket (\alpha = \beta) \rrbracket^{M,g} = \begin{cases} 1 & \text{if } I^g(\alpha) = I^g(\beta) \\ 0 & \text{otherwise} \end{cases} \quad (3.5)$$

$$\llbracket \neg\phi \rrbracket^{M,g} = \begin{cases} 1 & \text{if } \llbracket \phi \rrbracket^{M,g} = 0 \\ 0 & \text{otherwise} \end{cases} \quad (3.6)$$

$$\llbracket (\phi \wedge \psi) \rrbracket^{M,g} = \begin{cases} 1 & \text{if } \llbracket \phi \rrbracket^{M,g} = 1 \text{ and } \llbracket \psi \rrbracket^{M,g} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (3.7)$$

$$\llbracket (\phi \vee \psi) \rrbracket^{M,g} = \begin{cases} 1 & \text{if } \llbracket \phi \rrbracket^{M,g} = 1 \text{ or } \llbracket \psi \rrbracket^{M,g} = 1 \text{ (or both)} \\ 0 & \text{otherwise} \end{cases} \quad (3.8)$$

$$\llbracket (\phi \rightarrow \psi) \rrbracket^{M,g} = \begin{cases} 1 & \text{if } \llbracket \phi \rrbracket^{M,g} = 0 \text{ or } \llbracket \psi \rrbracket^{M,g} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (3.9)$$

$$\llbracket (\phi \leftrightarrow \psi) \rrbracket^{M,g} = \begin{cases} 1 & \text{if } \llbracket \phi \rrbracket^{M,g} = 0 \text{ and } \llbracket \psi \rrbracket^{M,g} = 0, \\ & \text{or } \llbracket \phi \rrbracket^{M,g} = 1 \text{ and } \llbracket \psi \rrbracket^{M,g} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (3.10)$$

$$\llbracket \exists x\phi \rrbracket^{M,g} = \begin{cases} 1 & \text{if there is a } h \approx_x g \text{ such that } \llbracket \phi \rrbracket^{M,h} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (3.11)$$

⁴ It is common in linguistics to use $\llbracket \cdot \rrbracket$ for the evaluation function (or interpretation in general), because most general semanticists use higher-order logic and $\llbracket \cdot \rrbracket$ has traditionally been used for higher-order logic.

$$\llbracket \forall x \phi \rrbracket^{M, g} = \begin{cases} 1 & \text{if for all } h \approx_x g \text{ it is the case that} \\ & \llbracket \phi \rrbracket^{M, h} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (3.12)$$

3.2.3 Explanation of the Rules for Predication and Quantification

As you can see, there is a reason why the ‘otherwise’ alternative is usually left out; it’s always the same. What do the clauses in the above definition mean? First of all, notice that the only new clauses are (3.4), (3.5), (3.11), and (3.12). The truth-functional connectives have exactly the same interpretation as in propositional calculus, and in this sense first-order predicate logic is an extension of propositional logic. What about the new rules then?

The predication rule (3.4) simply states that a predication of the form $P(a_1, \dots, a_n)$ is true if the n -tuple consisting of the interpretations of a_1 to a_n is an element of the interpretation of P , and P is interpreted as a set of n -tuples over the domain D_e , of course. So any n -ary predicate is defined in a fully extensional way by specifying a set of n -tuples when a model is specified. One thing to note about (3.4) is that the case of a unary predicate is sort of special: There is no 1-tuple. The rule must be understood in such a way that $P(a)$ is true if $I^g(a) \in I^g(P)$. This is a common convention.

Rule (3.5) ensures that the identity sign is interpreted as identity. Bear in mind that despite the infix notation the identity sign is an ordinary predicate and could have been written $=(a, b)$. However, a special rule is needed to ensure its proper semantics, because identity is strictly-speaking not first-order definable, i.e. it cannot be defined as an abbreviation on the basis of the quantifiers and the other rules in a first-order setting. A correct definition of identity requires second-order quantification over predicates as in the following formula:

$$\forall P \forall xy. (Px \leftrightarrow Py) \rightarrow (x = y) \quad (3.13)$$

This formula goes back to Leibniz (*Discourse on Metaphysics*, Section 9) and is called the *identity of indiscernibles*. The converse formula

$$\forall P \forall xy. (x = y) \rightarrow (Px \leftrightarrow Py) \quad (3.14)$$

is called the *indiscernibility of identicals*. The corresponding biconditional


$$\forall P \forall xy. (Px \leftrightarrow Py) \leftrightarrow (x = y) \quad (3.15)$$

is often called *Leibniz’ Law*. Neither of them are wffs of first-order predicate logic. Therefore, rule (3.5) is required, whereas we already know that we could do without explicit rules for most of the truth-functional connectives once a base like the Sheffer-stroke has been chosen.

The rules (3.11) and (3.12) quantify (in the meta language) over variants of assignments. This is one way of formally expressing variable binding. Take

for example an evaluation of the formula $\exists y.Py$ in a model M with respect to an assignment g . This wff is true if there is an y -variant h of g such that $\llbracket Py \rrbracket^{M,h} = 1$, which is the case if and only if $I^h(y) \in I^h(P)$, which is in turn the case if and only if $h(y) \in I^h(P)$. Since assignments are defined relative to the domain D_e of a model, the clauses effectively quantify over the objects in D_e while binding the respective variable.

3.2.4 Exercises

 **Exercise 30** Write definitions as in 3.4–3.12 for the following truth-functions and quantifiers:

- a. the Sheffer stroke
- b. the Peirce stroke
- c. negated conditional (i.e. corresponding to $\neg(\phi \rightarrow \psi)$)
- d. negated biconditional (i.e. corresponding to $\neg(\phi \leftrightarrow \psi)$)
- e. $\exists! x\phi$ with reading *there is exactly one x such that ϕ*
- f. $\exists_3 x\phi$ with reading *there are exactly 3 x such that ϕ*

3.3 Proof Theory

3.3.1 Tableaux Rules for First-Order Predicate Logic

Since the truth-functional connectives work exactly the same as in propositional logic the tableaux rules from propositional logic can be used in first-order logic with only minor changes. As far as the truth-functional connectives are concerned, theorems from propositional calculus are also theorems of first-order predicate logic. But of course additional rules for the quantifiers are needed. Table 3.1 depicts the tableaux rules for first-order predicate logic, where I have more or less copy & pasted the corresponding rules from the last chapter.

In the above rules, $\phi(x)$ stands for any wff in which x occurs freely one or more times and $\phi[x/c]$ is the same formula as ϕ except that all free occurrences of x in ϕ have been replaced by c . For example, Px and $\forall y.P(x,a) \rightarrow (Qx \wedge R(y,x))$ are formulas in which x occurs freely; thus, $Px[x/a] = Pa$ and $\forall y.P(x,a) \rightarrow (Qx \wedge R(y,x))[x/c] = \forall y.P(c,a) \rightarrow (Qc \wedge R(y,c))$.⁵ Notice that in the last formula we could only have replaced x by a (instead of c) if the corresponding rule was a universal quantification rule, because the constant a already occurs in the original formula.

⁵ Since square brackets are allowed as a substitute for ‘(’ and ‘)’ for better readability, they are now used for two different purposes. But it shouldn’t be too hard to differentiate between the two different usages.

Rules of propositional calculus:

$$\begin{array}{c} \phi \wedge \psi \\ | \\ \phi \\ \psi \end{array}$$

$$\begin{array}{c} \neg(\phi \vee \psi) \\ | \\ \neg\phi \\ \neg\psi \end{array}$$

$$\begin{array}{c} \phi \vee \psi \\ \wedge \\ \phi \quad \psi \end{array}$$

$$\begin{array}{c} \neg(\phi \wedge \psi) \\ \wedge \\ \neg\phi \quad \neg\psi \end{array}$$

$$\begin{array}{c} \neg\neg\phi \\ | \\ \phi \end{array}$$

$$\begin{array}{c} \phi \rightarrow \psi \\ \wedge \\ \neg\phi \quad \psi \end{array}$$

$$\begin{array}{c} \neg(\phi \rightarrow \psi) \\ | \\ \phi \\ \neg\psi \end{array}$$

$$\begin{array}{c} \phi \leftrightarrow \psi \\ \wedge \\ \phi \quad \neg\phi \\ \psi \quad \neg\psi \end{array}$$

$$\begin{array}{c} \neg(\phi \leftrightarrow \psi) \\ \wedge \\ \phi \quad \neg\phi \\ \neg\psi \quad \psi \end{array}$$

Existential quantification rules where constant c must be new on the branch:

$$\begin{array}{c} \exists x.\phi(x) \\ | \\ \phi[x/c] \end{array}$$

$$\begin{array}{c} \neg\forall x.\phi(x) \\ | \\ \neg\phi[x/c] \end{array}$$

Universal quantification rules where t is any ground term:

$$\begin{array}{c} \forall x.\phi(x) \\ | \\ \phi[x/t] \end{array}$$

$$\begin{array}{c} \neg\exists x.\phi(x) \\ | \\ \neg\phi[x/t] \end{array}$$

Table 3.1: Tableaux rules for first-order predicate logic without identity.

As you can see from the table, only the rules for the quantifiers are new. How do they work? Let's start with existential quantification. When there is a quantified wff on a branch, say $\exists x.Px$, then we can eliminate the quantifier by introducing a constant but this constant must be new to the branch. Suppose it wasn't new to the branch. Then we could, for example, derive $a \neq a$ from the claim $\exists x\exists y.x \neq y$ – but the former is a contradiction whereas the latter is clearly satisfiable! We know that some object, say a , exists such that Pa if $\exists x.Px$ is true, but $\exists x.Px$ does not allow us to make any additional assumptions about a . In order not to introduce any unwarranted assumptions, we are only allowed to introduce a new constant – one about which nothing has been said yet on the same branch – when simplifying $\exists x.Px$ in a truth-preserving way. The same applies for the negated universal quantifier. Why is this so? Take for example the statement 'not all students hate logic'. How would you confirm that this claim is true? The answer is fairly obvious. Find a student (in the domain of the model) who doesn't hate logic. So it seems that saying 'not all students hate logic' means just the same as saying 'there is a student who doesn't hate logic'. As this consideration shows $\neg\forall$ behaves like an existential quantifier, or to put it in other terms, a negated universal quantifier actually gives rise to an instance of existential quantification. This explains why the rule for $\neg\forall$ is analogous to the one for \exists and requires an individual constant that is new to the branch.

In contrast to this, the rule for the universal quantifier \forall allows you to introduce any constant for the fairly obvious reason that for example $\forall x.Px$ says that any object in the domain satisfies P – including the objects about which some other claims have been made already. Analogously to the previous case of existential quantification, the negated existential quantifier actually gives rise to an instance of universal quantification. To see this, take a correct literal paraphrasing of a wff involving a negated existential quantifier. Such paraphrases sound rather clumsy. Consider for example 'it is not the case that there is a student who failed the exam'. Speaking less like Mr. Spock this could be more adequately expressed as 'all students passed the exam'. This consideration intuitively explains why the rule for $\neg\exists$ allows choosing any constant as a replacement for the variable when the negated quantifier is eliminated in order to simplify the formula. Given a statement such as $\neg\exists x.Px$ we pick any individual named by a constant, say a , in our language we like and transform the statement into the claim that a does not satisfy P : $\neg Pa$. Negated existentials are hidden instances of universal quantification. The rules for \forall and $\neg\exists$ not only place no restriction on the choice of the constant, their application may also be repeated as often as you like. After all we're dealing with *universal* quantification and, surely, if $\forall x.Px$ then Pa holds, Pb holds, Pc holds, and so forth for all objects in the domain. From the fact that the universal rules may be repeated as often as one likes it follows that a tree might never be completed if one or more of its branches contain one or more instances of the unnegated universal or the negated existential quantifier and, in addition,

there is an infinite supply of constants naming objects in the domain D_e . To put it in other words, when each object in D_e has a unique name and the D_e has infinitely many members, then the tree cannot be completed.

Notice finally that the rules for universal quantification not only allow substitution by constants, but also substitution by functions that only take constants as arguments – for instance, $g(a)$ or $f(c, d)$. Why can they be substituted and why aren't they allowed in the rules for the existential quantifiers? Take a look at the definition of the interpretation function $I(\cdot)$ for functions in the beginning of section 3.2.2. A function symbol is interpreted as a function from elements in D_e , i.e. from $D_e \times \cdots \times D_e$, to one element in D_e . So if for example $\forall xPx$ is assumed to hold then $P(f(a, b, c))$ must also hold, because we know that $I^g(f(a, b, c))$ denotes an object in the domain D_e . However, in a rule for existential quantification we cannot assume that there are enough objects to satisfy the existential claim if we would substitute a grounded function instead of a constant only, because we could no longer check that the side condition is fulfilled. Recall that $\exists x\phi$ has a reading 'there are one or more x such that ϕ (of x)'. There might only be one such object. Suppose we have $\exists xPx$ on a branch. If we were allowed to obtain $P(f(a))$ and later perhaps $P(g(a))$ this would be unwarranted, because $f(a)$ and $g(b)$ might denote different entities but the original wff did not state that there are two objects that satisfy P . We could allow substitution by *one* ground term *once* on the branch but that would be unnecessary, because the result of that function could just be named by a constant c and we could use that constant instead. So it makes sense to only allow substitution by a constant that is new on the branch in case of the rules for existential quantification.

3.3.2 Rules for Identity

According to 3.5 the symbol '=' is interpreted as identity in the meta-language, but the above tableaux rules don't specify any special way how to deal with identity statements. Clearly, identity is not just like any other predicate. In an axiomatic setting one would add axiom schemes that express the logical properties of identity, i.e. transitivity, symmetry, and reflexivity because identity is an equivalence relation. For the tableaux we must add rules that allow us to take advantage of an identity statement once it has occurred on the branch. If for example $a = b$ occurs on the branch and somewhere else on the same branch $Pa \wedge \neg \forall x[R(a, x)]$ occurs on the branch, we should be allowed to write $Pb \wedge \neg \forall x[R(b, x)]$ instead. Moreover, we know that $t \neq t$ is a contradiction for any ground term t ; so if this occurs on a branch, the branch closes. Table 3.2 is the additional rule we need.

For any ground terms t, u :

$$\begin{array}{ccc}
 \phi & & \phi \\
 t = u & & t = u \\
 | & & | \\
 \phi[t/u] & & \phi[u/t]
 \end{array}$$

Table 3.2: Tableaux rule for identity.

3.3.3 Using the Tableaux Rules

The tableaux rules are used in the same way as those for propositional logic with one important difference. Because of the rule for universal quantifications, i.e. the rules for \forall and $\neg\exists$, the tree might never be completed. Some care is needed to substitute the right constant when applying one of the universal rules. Usually, this will be the same constant as was already used before when applying one of the existential rules. Because of the restriction for the existential rules, i.e. the rules for \exists and $\neg\forall$, it is generally a good strategy to first apply an existential rule and then apply a universal rule whenever there is a choice between the order of two such rule applications. Otherwise it might be necessary to apply the same universal rule twice in order to close the branch. When there are two universal quantifications, it is often a good strategy to use the same constant for simplifying both of them. Here are a few example proofs.

*** Example 7** To show: $\forall xPx \rightarrow \exists xPx$. Proof: We start with the negated wff. Recall that the formula could also be written as $\forall x[Px] \rightarrow \exists x[Px]$. The scope of \forall is narrow and the main junctor is \rightarrow . So we apply the rule for $\neg\rightarrow$ and then the universal rules for \forall and $\neg\exists$ with the same constant.

$$\begin{array}{c}
 \neg(\forall xPx \rightarrow \exists xPx) \\
 | \\
 \forall x.Px \\
 \neg\exists xPx \\
 | \\
 Pa \\
 \neg Pa
 \end{array}$$

The tree closes, hence $\forall xPx \rightarrow \exists xPx$ is a tautology. QED.

*** Example 8** To show: $\exists x \neg Px \rightarrow \neg \forall x Px$. Proof: The main junctor of the formula to prove is \rightarrow . We negate the formula and use tableaux rules:

1.	$\neg(\exists x[\neg Px] \rightarrow \neg \forall x[Px])$	assumption
2.	$\exists x. \neg Px$	1: $\neg \rightarrow$
3.	$\neg \neg \forall x. Px$	1: $\neg \rightarrow$
4.	$\forall x. Px$	3: $\neg \neg$
5.	$\neg Pa$	2: \exists
6.	Pa	4: \forall

The tree closes with 5 and 6, hence $\exists x \neg Px \rightarrow \neg \forall x Px$ is a tautology. QED.

Notice that it would *not* have been correct to first use \forall and afterward \exists , because the rule for \exists would have required us to use a new constant instead of a ! More examples can be found in the next section.

Hints. Remember the following points:

- If you don't know whether it is a theorem or not, quickly assess the original formula before using the proof method: Should this hold? Can I find a counterexample beforehand?
- Whenever possible, use existential rules \exists and $\neg \forall$ first, universal rules afterward.
- Use a previously used constant when applying a universal rule \forall or $\neg \forall$.
- Be prepared to apply a universal rule several times.
- Never ignore the side condition of the existential rules: You need to use a new constant that does not yet occur on the same branch – no exceptions!

3.3.4 Selected Theorems

T1. $\forall x Px \rightarrow \exists x Px$

Proof: see above.

T2. $\forall xy R(x, y) \leftrightarrow \forall yx R(x, y)$

T3. $\forall x [Px \rightarrow \exists y Qy] \leftrightarrow \forall x \exists y [Px \rightarrow Qy]$

T4. $\neg\exists x[x \neq x]$

Proof:

$$\begin{array}{c} \neg\neg\exists x[x \neq x] \\ | \\ \exists x[x \neq x] \\ | \\ a \neq a \end{array}$$

The tree closes, because $a \neq a$ is a contradiction. QED.

T5. $\forall x[Px \rightarrow Qx] \rightarrow (\forall x[Px] \rightarrow \forall x[Qx])$

T6. $\exists xPx \rightarrow \neg\forall x\neg Px$

Proof:

$$\begin{array}{c} \neg(\exists xPx \rightarrow \neg\forall x\neg Px) \\ | \\ \exists xPx \\ \neg\neg\forall x\neg Px \\ | \\ \forall x\neg Px \\ | \\ Pa \\ | \\ \neg Pa \end{array}$$

The tree closes. QED.

T7. $\forall x\exists y.Px \rightarrow Py$

Proof:

$$\begin{array}{c} \neg\forall x\exists y[Px \rightarrow Py] \\ | \\ \neg\exists y[Pa \rightarrow Py] \\ | \\ \neg[Pa \rightarrow Pa] \\ | \\ Pa \\ | \\ \neg Pa \end{array}$$

The tree closes. QED.

T8. $\forall x.Px \rightarrow \exists xPx$

T9. $\forall x[Px \wedge Qx] \leftrightarrow \forall x[Px] \wedge \forall x[Qx]$

T10. $[\forall x(Px) \vee \forall x(Qx)] \rightarrow [\forall x(Px \vee Qx)]$

Proof:

$$\begin{array}{c}
 \neg([\forall x(Px) \vee \forall x(Qx)] \rightarrow [\forall x(Px \vee Qx)]) \\
 | \\
 \forall x(Px) \vee \forall x(Qx) \\
 \neg[\forall x(Px \vee Qx)] \\
 | \\
 \neg(Pa \vee Qa) \\
 | \\
 \neg Pa \\
 \neg Qa \\
 | \\
 Pa \\
 Qa
 \end{array}$$

The single branch of the tree closes. QED.

T11. $\exists x[Px \vee Qx] \leftrightarrow \exists x[Px] \vee \exists x[Qx]$


T12. $\exists x[Px \wedge Qx] \rightarrow [\exists x(Px) \wedge \exists x(Qx)]$

Proof:


$$\begin{array}{c}
 \neg(\exists x[Px \wedge Qx] \rightarrow [\exists x(Px) \wedge \exists x(Qx)]) \\
 | \\
 \exists x[Px \wedge Qx] \\
 \neg[\exists x(Px) \wedge \exists x(Qx)] \\
 | \\
 Pa \wedge Qa \\
 | \\
 Pa \\
 Qa \\
 \swarrow \quad \searrow \\
 \neg\exists xPx \quad \neg\exists xQx \\
 | \quad \quad | \\
 \neg Pa \quad \quad \neg Qa
 \end{array}$$

Both branches of the tree close. QED.


3.3.5 Exercises

 **Exercise 31** Prove the following theorems.

- | | | |
|-------|-------|--------|
| a. T2 | c. T5 | e. T9 |
| b. T3 | d. T8 | f. T11 |

 **Exercise 32** Check, using semantic tableaux, whether the following formulas are valid or not.

- a. $Px \vee \neg Px$

 **Exercise 33** Prove that the following holds.

a. (Hodges 1977) Every irreflexive and transitive binary relation is asymmetric:

$$\forall x. \neg Rxx, \forall xyz. [Rxy \wedge Ryz] \rightarrow Rxz \vdash \forall xy. Rxy \rightarrow \neg Ryx$$

3.4 Defined Notions

3.4.1 Russellian Descriptions

Going back to work by Russell, the iota operator is a term-building operator that takes a formula and yields the unique object that satisfies this formula. A term $\iota x.Px$ is read as *the x such that P of x*. It can be defined as follows.

$$\text{Syntax: If } \phi \in L_S \text{ and } x \in L_V \text{ then } \iota x\phi \in L_T \quad (3.16)$$

$$\text{Semantics: } I^g(\iota x\phi) = \begin{cases} h(x) \text{ if there is exactly one } h \approx_x g \\ \text{such that } \llbracket \phi \rrbracket^h = 1, \\ \text{undefined otherwise} \end{cases} \quad (3.17)$$

However, this makes $I(\cdot)$ a partial function from terms to denotations. If there are more objects that satisfy ϕ or there is no object satisfying ϕ , then $I^g(\iota x\phi)$ is undefined. Consequently, rule 3.4 on page 72 would have to be adjusted as follows.

$$\llbracket P(\alpha_1, \dots, \alpha_n) \rrbracket^{M,g} = \begin{cases} 1 \text{ if all of } I^g(\alpha_1), \dots, I^g(\alpha_n) \text{ are defined} \\ \text{and } \langle I^g(\alpha_1), \dots, I^g(\alpha_n) \rangle \in I^g(P), \\ 0 \text{ otherwise} \end{cases} \quad (3.18)$$

There is a way to achieve the same effect as with the iota operator but without resorting to partial interpretations. We can define the following two-place quantifier.

$$\iota x[\phi]\psi := \exists x[\phi \wedge \forall y(\phi[x/y] \rightarrow x = y) \wedge \psi] \quad (3.19)$$

This can be read as *the x that uniquely satisfies ϕ also satisfies ψ* or in a similar way.⁶ The formulas $P(\iota x.Qx)$ and $\iota x[Qx]Px$ are equivalent in any model, and so the quantifier can replace uses of the iota operator in predicative clauses. Strange enough, there doesn't seem to be any common name for this two-place quantifier even though it is well-known since Russell's times. I have used and will continue to use the term 'iota quantifier' for it.

⁶ One advantage of the iota operator over the quantifier is that it is easier to paraphrase. It is hard to express unambiguously by the paraphrase that in a use of the above quantifier the uniqueness condition is only put on ϕ , not on ψ .

☞ **Note 11 (Definability, Characterizability.)** When an expression such as the iota quantifier is definable as an abbreviation like 3.19 in first-order logic, we say that it is *first-order definable*. Generally, proofs that an expression with a certain semantics can be defined with the means of a given logical system are called *characterization results* and play an important role in logical research, because they are one way to circumscribe the *expressive power* of a formal system. Not all natural language quantifiers are first-order definable. For example, the quantifier ‘most’, whose definition was given in set-theoretic terms in note 3 (on page 7 of chapter 1), is not first-order definable.

3.4.2 Relativized Quantifiers

It is possible to define *relativized quantifiers*, which are restricted to a certain domain of objects. Let for example $D(x)$ be a special domain predicate. Relativized quantifiers could then be defined as:⁷

$$\forall^* x\phi := \forall x[Dx \rightarrow \phi] \quad (3.20)$$

$$\exists^* x\phi := \exists x[Dx \wedge \phi] \quad (3.21)$$

D is a quantifier domain restriction for the quantifiers. This restriction could for example be regarded as being contextually provided in order to get a more reasonable account of natural language quantification. For one may argue that an utterance like 3.22 doesn’t mean that every student on earth is in the classroom and it is hard to imagine a context in which an utterance of 3.23 is meant to be read as *every bottle in the universe is empty*.

(3.22) Every student is in the classroom.

(3.23) Every bottle is empty.

However, we are now speaking of utterances instead of sentences and no specific contextual resolution mechanism is provided by an analysis based on a simple domain predicate. The status of quantifier domain restrictions and similar phenomenas regarding the semantics/pragmatics distinction and the correct way of modeling them on the basis of a finite lexicon and grammar are still subject to philosophical debates. Nevertheless it is commonly assumed in linguistics that natural language quantifiers are contextually restricted. Using relativized quantifiers is one means of achieving this.

⁷ Why the definition for the universal quantifier looks different from the one of the existential quantifier will be explained in detail further below.

3.4.3 Many-sorted Logic

Instead of one domain D_e for objects you could introduce another domain, say D_s for situations, add variables s, s_1, s_2, \dots, s for situations and an extra pair of quantifiers that take situation-variables and run over D_s . This implementation of first-order logic would be *two-sorted*. In the same manner any kind of other extra domains could be added, making the logic *many-sorted*.

While having many different variables with corresponding quantifiers can be handy from a notational point of view, it does not add anything to the expressivity of the logic in general. Instead, you may introduce a domain predicate into the object language for each sort and define corresponding restricted quantifiers by abbreviation. Whether many-sorted first order logic (for finitely many sorts) or relativized quantifiers are used is merely a matter of convenience.

3.4.4 The Existence Predicate

Nothing in the formulation of first-order logic prevents us from introducing an existence predicate and possibly also define restricted quantifiers on the basis of it just like in 3.20 and 3.21. Let $E(x)$ be such a unary existence predicate with intended interpretation x exists. We can then represent 3.24 as 3.25, where a is a constant for Santa Claus and $P(x)$ a predicate with reading x has a long, white beard.

(3.24) Santa Claus doesn't exist but he has a long, white beard.

(3.25) $\neg E a \wedge P a$

Using the relativized quantifiers we may quantify over existing objects, whereas the unrestricted quantifiers run over all objects.

There is a long philosophical tradition of criticizing such uses of an existence predicate. Some logicians have a strong philosophical conviction that *actualism* is the right position towards non-existence: We can only ascribe (positive) properties to things that exist. In contrast to this, allowing meaningful talk about nonexistent objects is a decidedly *possibilist* position. One might ask how we can know which properties a particular nonexistent object has. This concern is legitimate and many different answers have been given to it, but at one point one point or another one has to opt for a possibilist position if one is interested in natural language semantics. It is pointless to claim, from a fundamentalist point of view, that 3.24 cannot be true despite the fact that we commonly regard it as being true. Theories of fictional objects deal with this problem.

Another concern with the existence predicate is based on the idea that unsound proofs like the ontological proof for the existence of God mentioned in section 2.4 of the last chapter would go through if existence were a predicate. Despite its persistence this criticism is not justified. First, the fact that the

conclusion of a valid argument that is intuitively judged sound might be undesirable or appear to be implausible doesn't for itself suffice for challenging any of its premises, although we sometimes take an implausible conclusion of a valid argument as an indicator that something must be wrong with one of its premises. The mere conclusion that God exists doesn't seem to suffice to turn the argument into a *reductio ad absurdum*. From a logical point of view, in a *reductio ad absurdum* we merely assume the premises without considering them plausible and then derive a contradiction, whereas in this case some people might merely find the conclusion somewhat implausible (depending on their beliefs). Second, it is an open question whether the premises of (particular versions of) the ontological argument are sound. Saying that the most supreme being must exist, because not to exist is a flaw, is not substantially different from saying that Erich's best toaster must exist, because a nonexistent toaster cannot be the best toaster. This premise is not true and, moreover, I don't have a toaster. But if the premises of the ontological argument are not sound anyway, then it doesn't matter whether they establish the conclusion or not. Third, 'to exist' is a verb that in finite clauses occurs as a grammatical predicate; there is no linguistic evidence that it should be translated to anything else than a logical predicate.

Two uses of the existence predicate have to be distinguished: In a strict actualist setting the existence predicate must be reducible. 'Reducible' here means that a rule $Pt \vdash Et$ is added and $\vdash Et$ holds for any term t . This makes the existence predicate redundant, but it may still be used to analyze 'to exists'. In a possibilist setting, on the other hand, no restriction is put on the existence predicate and the existence predicate has no special logical properties. It is just another predicate with a certain intended meaning and simply divides the total domain D_e in two halves: the set of objects a for which $\llbracket Ex \rrbracket^{M.g[x/a]}$ is true and the set of objects for which it is false. This use of the existence predicate is not redundant but also does not change the logic, because in this use the existence predicate has no special logical properties. This use is exemplified by 3.25.

3.5 Applications to Natural Languages

3.5.1 Truth-Conditions and Pre-Montegovian Semantics

Before Richard Montague and others popularized higher-order logic and categorial grammar in linguistics at the end of the sixties and the beginning of the seventies of last century, sentence-level semantics was mostly based on the specification of truth-conditions of sentences in first-order logic. In fact, a large number of natural language expressions have first-order characterizable truth-conditions and it is possible to specify reasonable semantic representations of many natural language sentences in first-order predicate logic. The representations are less elegant than the ones in higher-order logic that will be discussed in the next chapter and it is very hard to provide a mechanical map-

ping from natural language to first-order logic, because many natural language constructions can only be expressed by ‘tricks’ in first-order logic. We will take a look at some examples and some of these ‘tricks’ in the following paragraphs.

The Main Verb. A main verb with n obligatory argument places can be represented by an n -ary relation. For example, ‘give’ may be represented by a predicate $P(x, y, z)$ with reading x gives y to z or ‘to buy’ may be represented by a predicate $P(x, y, z, z')$ with reading x buys y from z at the price z' .

Proper Names. A proper name can be represented by a constant, e.g. ‘Erich’ may be represented by a , ‘Maria’ by b , João Gonçalves by c , and so forth. Alternatively, one could use a unary predicate in combination with a iota operator or iota quantifier. For example, ‘João laughs’ could be expressed as $L(\iota x.Px)$ or $\iota x[Px]Lx$. This use of descriptions for proper names was advocated by Russell and later criticized by Saul Kripke in *Naming and Necessity* on the basis of philosophical and linguistic intuitions about how proper names are understood in modal claims such as ‘Aristotle might not have been the teacher of Alexander the Great’. There is an extensive literature about this topic and in the aftermath of Kripke’s work proper names are most commonly represented by constants.

Sentence Connectives. It is natural to translate ‘and’ and ‘but’ to \wedge , ‘or’ to \vee , ‘if... then...’ to \rightarrow and so on – as long as you keep in mind that there are many exceptions to these rules!

Quantifiers. ‘all’ and ‘every’ can be expressed by \forall and English ‘a’ by \exists . Some, though by far not all, uses of the English determiner ‘the’ can be represented by the iota quantifier or operator. Other quantifiers like ‘exactly three’ are also first-order definable. Others like ‘most’ cannot be expressed in first-order logic.

☞ **Note 12 (Correct Use of \exists and \forall)** As a general rule when using \forall it must be combined with the conditional in order to get the intended reading. For example, ‘all men are mortal’ becomes $\forall x[Men(x) \rightarrow Mortal(x)]$. The existential quantifier \exists , on the other hand, usually needs to be combined with conjunction in order to get the intended reading. For example, ‘there is a mortal man’ becomes $\exists x[Man(x) \wedge Mortal(x)]$.

Why is this so? It is clear in case of the existential quantifier that in order to be a mortal man you need to be mortal and a man. (This translation of adjectives does not work in general, though. See below.) But if we would use conjunction for ‘all men are mortal’ we would get a completely inadequate reading: $\forall x[Men(x) \wedge Mortal(x)]$ all things in the domain of the model are both men and mortal, which is clearly not what ‘all men are mortal’ means.

Recall from chapter 1 that set-theoretically ‘all men are mortal’ can be represented as $Men \subseteq Mortals$, i.e. the set of men is a subset of the set of mortal things. $\forall x[Man(x) \rightarrow Mortal(x)]$ encodes exactly the same truth conditions. Suppose $Men = \emptyset$. Then the above statement is true, because $\emptyset \subseteq A$ for any set A . Likewise, if $Man(x)$ turns out false, $Man(x) \rightarrow Mortal(x)$ is true. If on the other hand $Men \neq \emptyset$ then if $Men \subseteq Mortals$ holds, $Mortals \neq \emptyset$ must hold as well – because obviously a non-empty set cannot be a subset of the empty set. Likewise, if $\forall x[Man(x) \rightarrow Mortal(x)]$ and $\exists x.Man(x)$ (which could be read as ‘the set of men is non-empty’), then it is also the case that $\exists x.Mortal(x)$ (which could be read as ‘the set of mortal things is non-empty’).

Notice also the following equivalences and the role that negation plays in it:

$$\forall x[Px \rightarrow Qx] \leftrightarrow \neg \exists x \neg [Px \rightarrow Qx] \quad (3.26)$$

$$\leftrightarrow \neg \exists x \neg [\neg Px \vee Qx] \quad (3.27)$$

$$\leftrightarrow \neg \exists x [\neg \neg Px \wedge \neg Qx] \quad (3.28)$$

$$\leftrightarrow \neg \exists x [Px \wedge \neg Qx], \quad (3.29)$$

and

$$\exists x [Px \wedge Qx] \leftrightarrow \neg \forall x \neg [Px \wedge Qx] \quad (3.30)$$

$$\leftrightarrow \neg \forall x [\neg Px \vee \neg Qx] \quad (3.31)$$

$$\leftrightarrow \neg \forall x [Px \rightarrow \neg Qx]. \quad (3.32)$$

Adjectives. Some adjectives can be represented in a straightforward way while others are pretty hard to express out-of-the-box in first-order logic. For example, ‘red ball’ may be represented as $Rx \wedge Bx$ (R for redness and B for being a ball), but such a conjunctive analysis is inadequate for ‘famous pianist’. Why? Someone can be famous for his virtuosity with the sledgehammer and at the meantime be a lousy pianist, but this doesn’t make him a famous pianist.

Indexicals and Anaphora. As long as they stand for ordinary individuals, indexicals and anaphora can be expressed in first-order logic either as open variables or as functions from some suitable type of context ‘individual’ to an ordinary object. For example, ‘I’ in ‘I’m hungry’ could be represented as Hx with the convention that x represents the speaker of the utterance and ‘he’ in ‘John is hungry. He eats a banana’ could be represented as $Ha \wedge \exists x \exists y. By \wedge E(x, y) \wedge x = a$. In a more elaborate account, one could stipulate that contexts (whatever they are) are in D_e and express ‘I’m hungry’ as $\exists x.H(f(x))$, where f in the model represents a function taking an utterance situation and yielding the speaker

in that utterance situation. For a more convenient notation it would also be possible to introduce an additional domain D_c for contexts and make the logic two-sorted.

Tenses. Like in the case of adverbs tenses require the introduction of time intervals or situations, events, or other objects on which a temporal ordering relation is defined. When time intervals or other temporally ordered entities are available a denotational semantics for the basic absolute tenses can be implemented. Suppose we modify *FOL* and introduce a new sort of variables t , t_1 , t_2, \dots for time intervals with corresponding quantifiers \exists^t and \forall^t . Let $<$ be a temporal ordering of the domain of time intervals D_t in the meta language such that $t_1 < t_2$ if t_1 ends before t_2 starts. We can then add an argument place for a time interval to each predicate that corresponds to a finite verb and formulate rules for basic absolute tenses:

$$\llbracket Past(t_1, t_2) \rrbracket^{M,g} = \begin{cases} 1 & \text{if } g(t_1) < g(t_2) \\ 0 & \text{otherwise} \end{cases} \quad (3.33)$$

$$\llbracket Fut(t_1, t_2) \rrbracket^{M,g} = \begin{cases} 1 & \text{if } g(t_2) < g(t_1) \\ 0 & \text{otherwise} \end{cases} \quad (3.34)$$

Assuming a convention that the variable t_0 is interpreted as the time of utterance we may then express 3.35 ad 3.36 and 3.37 as 3.38.

(3.35) John was hungry.

(3.36) $\exists t_1. Past(t_1, t_0) \wedge Hungry(t_1, j)$

(3.37) John will meet Mary.

(3.38) $\exists t_1. Fut(t_1, t_0) \wedge Meet(t_1, j, m)$

For a more detailed treatment of tenses and aspect more relations between time intervals are needed. We need to be able to express that one time interval is a subinterval of another and when two time intervals overlap. These relations are first-order axiomatizable. See van Benthem (1991) *The Logic of Time* for more details.

3.5.2 Some Problems

Let me now give a few examples that illustrate the complexity of natural languages and that even simple sentences may elicit complicated semantic problems whose solution may be approached in different ways. There are numerous similar examples, of course, and the work of the general semanticist is to a large extent to classify different readings of natural language expressions and somehow determine how they might be represented in first-order logic, higher-order logic, or using other tools.

Belief and other Attitudes. It is very hard to correctly implement belief and similar attitudes, including the meaning of factive verbs such as ‘to know’, in out-of-the-box first-order predicate logic. They are usually implemented on the basis of modal logic and generally considered as so-called *intensional verbs*. Consider the following sentence:

(3.39) John believes that Mary is hungry.

The verb ‘to believe’ is clearly not truth-functional. The truth value of the whole sentence does not directly depend on the truth-value of the embedded sentence. It is possible to introduce an operator into first-order logic that builds a name for a wff. Suppose – for the sake of the argument and without going into the details – that $\ulcorner \phi \urcorner$ builds a constant for the formula ϕ . Then we can analyze belief as a predicate as in the following formula:

$$Bel(j, \ulcorner H(m) \urcorner) \quad (3.40)$$

However, this way of analyzing attitudes is not very popular for a variety of reasons. First, it relates John to a formula of a specific formal language. But why? Perhaps John doesn’t know anything about logic, but the relation is really stipulated to hold between him and *the formula itself*. Shouldn’t such a belief relation rather be a relation between John and what the formula means? The formula itself is just a string of symbols! Second, a belief predicate of this kind only works as long as no strong *introspection principles* are assumed. Such principles do, for example, assert that whenever someone believes that ϕ , he also believes that he believes that ϕ (positive introspection) or that whenever someone does not believe that ϕ he believes that he does not believe that ϕ (negative introspection). Another such principle is for example needed for the factivity of ‘to know’, namely that if someone knows that ϕ , then ϕ is the case (KK principle or factivity of knowledge). These introspection principles are available in different versions of modal logic and can be integrated into first-order predicate logic by embedding modal logics into it. The resulting systems are known under the label *first-order modal logic*.

Montague (1974) showed that when a belief predicate is added to first-order predicate logic, a way of quantifying over embedded propositions is added (as in ‘Everything John believes is false’ $\forall p[Bel(j, p) \rightarrow \neg p]$), and sufficiently strong introspection principles are assumed, then the logic becomes inconsistent.⁸ These problems can be circumvented with some fairly technical trickery and then the syntactic treatment of belief is more expressive than the modal logical one (see Bolander 2003). Nevertheless, the usual, established way to deal with so-called *propositional attitudes* like belief is to use a system of modal logic (often one called KD45) or more elaborate approaches that have descended from modal

⁸ Montague, R. (1974). Syntactical Treatments of Modality, with Corollaries on Reflexion Principles and Finite Axiomatizability Formal Philosophy. In Thomason, R. (Eds.) (1974). *Selected Papers of Richard Montague*, (pp. 287–302). Yale University Press.

logic. The bottomline of this section is that you ought not attempt to analyze belief, knowing that, and other attitudes that take an embedded sentence as complement (e.g. ‘to fear’, ‘to doubt’) as predicates unless you know what you’re doing.

Asymmetric Conjunction. Asymmetric conjunction is a special reading of a use of the seemingly innocuous word ‘and’ at sentence level. Consider the following utterances:

(3.41) Lea grabbed the bottle of whiskey and took a big sip.

(3.42) Lea took a big sip and grabbed the bottle whiskey.

Speakers tend to read 3.41 such that Lea *first* grabbed the bottle of whiskey and *then* took a big sip *out of that bottle of whiskey*, whereas this interpretation is not so readily available in 3.42. Traditional truth-conditional semanticists would consider this a pragmatic phenomenon as in the contrastive reading expressed by ‘but’. On the other hand, it is pretty hard to interpret 3.41 in a way such that Lea took a big sip out of a completely different bottle.

Adverbs. Without introducing contexts, situations, events, or possible worlds specifying an adequate truth-conditional meaning for adverbs is difficult or even impossible in classical first-order logic. Sentence adverbs like ‘presumably’ or ‘timely’ have a complicated meaning and modify the meaning of the sentence as a whole just like tenses and moods. Even adverbs that only modify a verb or adjective are hard to implement in first-order logic, because it doesn’t allow us to quantify over predicates or express a function from a predicate to a new predicate. Consider for example the intensifier ‘very’ in ‘Maria walks very fast’. At least *prima facie* it seems that ‘very’ modifies the meaning of ‘fast’ – and it would be mind-boggling to think of this as a conjunctive condition, because there are no ‘very’ things or events.

Counterfactual Conditionals Consider counterfactual conditionals like the following one.

(3.43) If Kennedy had pressed the button, the world would have been destroyed in a nuclear Armageddon.

This sentence cannot be represented by an ordinary truth-functional conditional, because it involves certain deliberations about counterfactual scenarios. Many different non-classical logics for counterfactual conditionals have been proposed

and there is not much of an agreement about what exactly the truth-conditions of sentences like 3.43 are and whether they are first-order definable or not.⁹

NP-Conjunction. Consider the following uses of ‘and’ for NP-conjunction:

(3.44) Ontem Maria e João foram no cinema.

(3.45) John and Peter dragged the washing machine six floors over the stairs to my apartment.

Ordinary truth-functional conjunction in first-order logic only allows us to combine sentences with sentences. But even if we ignore for a moment how to obtain it in a systematic way from the natural language sentence, does 3.44 even allow a reading as *Yesterday Maria went to the cinema and yesterday João went to the cinema (but not necessarily the same and together)*? Can it express a stronger condition, saying *Yesterday Maria and João went to the cinema together*, and if so, how can this be represented in FOL? Moreover, anyone who has ever carried a washing machine will be aware that only the stronger, second reading can be present in 3.45. But how is this reading triggered and is it part of the truth-conditional meaning of the sentence?

3.5.3 Deductive Arguments

Deductive arguments are represented and defined the same way as in propositional logic except that the first-order quantifiers are now available. For example, we can now represent the following classical argument.

(3.46) All men are mortal.

(3.47) Socrates is a man.

(3.48) Hence, Socrates is mortal.


The translation to first-order predicate logic is: $(\forall x[Px \rightarrow Qx] \wedge Pa) \rightarrow Qa$. Using first-order tableaux we can prove the validity of the argument by assuming the premises in the antecedent of the main conditional, denying the consequent, and checking that the tree closes.

⁹ Bear in mind that first-order definability is mostly a matter of the quantification involved. In one account for counterfactuals 3.43 would be true if in the most plausible scenarios in which Kennedy has pressed the button is a nuclear Armageddon.


$$\begin{array}{c}
 \forall x[Px \rightarrow Qx] \\
 Pa \\
 \neg Qa \\
 | \\
 Pa \rightarrow Qa \\
 \wedge \\
 \neg Pa \quad Qa
 \end{array}$$

The tree closes, hence the argument is valid. Again, the soundness of the premises must be established in order to make the argument convincing at all. Notice that this task is particularly difficult when universal quantification is at play, i.e. either \forall or $\neg\exists$ are used. While from a strictly logical point of view one counterexample to a universal claim suffices to disprove the universal claim it is in many real-world cases almost impossible to show that no such counterexample exists. That is the reason why fallibilists like Karl Popper have emphasized that empirical scientific theories – which usually involve universal statements – cannot be verified once and for all. They can only be confirmed by positive evidence and perhaps later disproved by counterexamples. From a practical perspective, this is a good rule of thumb for many scientific theories even if the domain is restricted and ultimately finite. For example, a biologist might not be able to check whether animals of a certain species have a certain property and be absolutely certain that he didn't miss one, although there are only finitely many beings on earth. In many other cases, however, the domain is not only clearly finite but it is also practical to make use of this fact. For example, I can easily verify a universal claim about the coins in my wallet by exhaustively checking each one of them.

3.5.4 Exercises


 **Exercise 34** Using the predicates Sx (' x can solve this problem'), Mx (' x is a mathematician'), Jx (' x is Joe'), translate the following arguments into first-order predicate logic and check whether they are valid using semantic tableaux. (taken from Simpson (2004))

- If anyone can solve this problem, some mathematician can solve it. Joe is a mathematician and cannot solve it. Therefore, nobody can solve it.
- Any mathematician can solve this problem if anyone can. Joe is a mathematician and cannot solve it. Therefore, nobody can solve it.

 **Exercise 35** Define predicates and relations with adequate readings for each of the following sentences and formulate truth-conditions for at least one reading of each of them in first-order predicate logic. (Tenses and possessive pronouns like 'his' should be ignored.)


- Peter likes Mary even though she can't stand him.

- b. O Pedro tem um gato ou um cão.
- c. Either Ana or Maria has a car.
- d. Every sailor has a ship.
- e. All Cretans except Epimenides are liars.
- f. Everything Midas touches turns into gold.
- g. O Pedro dá o seu livro à Ana.
- h. Anyone who counterfeits this \$10 bill has to pay a fine or go to prison.
- i. It was the gardener or the butler who killed Lady Buttersworth. If it was the gardener then the housemaid was an accomplice. If it was the butler then the taxi driver must have seen him. The taxi driver did not see the butler. Hence, Lady Buttersworth was killed by the gardener and the housemaid was an accomplice.
- j. Not everything made of gold shines.
- k. O Afonso gosta de peixe porque peixe é saudável.
- l. No student deserves a beer unless he has finished his logic exercises.
- m. Cada professor tem um carro mas há estudantes que não têm um.
- n. Se este livro não conter um erro os estudantes o encontram.

 **Exercise 36** So called Donkey-sentences have been one of the main motivations for developing dynamic predicate logic (DPL). Here is such a sentence:

(3.49) Every farmer who owns a donkey beats it.

Provide a wff that expresses the correct truth-conditions for (at least) one reading of the sentence.

 **Exercise 37** This is one of my favorite ‘joke’ proofs:

Nothing is better than a steak.
A salad is better than nothing.

A salad is better than a steak.

- a. What is the main mistake in the argument? Why the conclusion does not follow?
- b. Formalize the alleged proof and prove that the result is contradictory.

✎ **Exercise 38** Translate the following deductive argumentation fragments into propositional logic, idealizing the statements as is deemed appropriate, and determine whether they are deductively valid or based on a fallacy.

- a. Anyone who eats animals is evil. If someone is a vegetarian then he does not eat animals. John is not a vegetarian or he just accidentally ate a big, yummy steak. Therefore, John is evil.
- b. Eating dolphins does not amuse fourteen year old girls. João just ate a dolphin and it amused Patricia. Therefore, Patricia is not a fourteen year old girl.
- c. Smoking marihuana is prohibited to anyone unless he has a painful medical condition. Jack does not have a painful medical condition. Therefore, Jack is not allowed to smoke marihuana.
- d. Every real American who owns a TV also owns a car. Everyone who owns a car earns more than \$ 100000 a year. The Jacksons don't make more than \$ 100000 a year and don't even own a TV. Hence, they are not real Americans.
- e. Every decent chap likes coffee or cigarettes. Jacky, a convicted mass-murderer and baby-eater, likes coffee and cigarettes. Therefore, Jacky is a decent chap.
- f. Mushrooms can cause hallucinations. Maria had scrambled eggs with mushrooms, orange juice, and a slice of toast for breakfast and is hallucinating. Therefore, the mushrooms caused her hallucinations.

3.6 Metatheorems

First-order predicate logic is sound, complete, and compact. However, first-order predicate logic is *undecidable*, or, more precisely, it is *semidecidable*. Being semidecidable means the following: While there is a terminating procedure for determining that a given formula is valid (if it is one), there is no procedure that determines in finite time that a given formula is not valid.¹⁰

According to the *Löwenheim-Skolem theorem* any satisfiable first-order formula is satisfiable in a countable domain, i.e. in a domain whose cardinality is not larger than \aleph_0 .¹¹ When this was first proved it was a matter of great concern to many logicians. In side note 1 (page 6, chapter 1) it was mentioned that the cardinality of the set of real numbers is higher than \aleph_0 . However you put it there

¹⁰ Being undecidable means that there is neither a terminating procedure for determining that a given formula is valid nor one for determining that it is not valid. Since in the case of a semidecidable logic for a formula whose status with respect to validity is unknown there is no procedure that is guaranteed to decide in finite time whether a given wff is valid or not – such a procedure might not halt – the difference between semi- and full decidability doesn't matter much and first-order logic is often just labeled as being undecidable.

¹¹ Obviously, it could be smaller in case the formula is satisfiable in a finite domain.

are ‘more’ real numbers than natural numbers, because you cannot create a bijection between \mathbb{N} and \mathbb{R} . Now it follows from the Löwenheim-Skolem theorem that any (consistent) arithmetic theory of real numbers formulated in first-order logic has a countable domain, i.e. all the formulas of the theory are satisfiable in a countable domain even though they are supposed to characterize the real numbers. (This is sometimes referred to as ‘Skolem’s paradox’.) First-order logic cannot distinguish between countable and uncountable domains.

3.7 Literature

Semantic tableaux are also sometimes called Smullyan calculi, because Raymond Smullyan pioneered them. He has written numerous books on logic including ones intended for a general audience. His logical puzzles have entertained generations of professionals and laymen.

- Raymond M. Smullyan (1995). *First-order Logic*. Dover.

Smullyan’s introduction is a classic and highly recommended. I have also already praised Hodges’ introduction in the last chapter, which is particularly well-suited for linguists.

- Wilfried Hodges (1977, 2001). *Logic: an introduction to elementary logic*. Penguin.

For people interested in a little bit more mathematical background information, the following two books are a good start. Ebbinghaus et. al. (1994) is a standard text aimed at people with a strong mathematical background interested in metatheorems and properties of classical logics. If you want to buy it, get the latest edition. Andrews (2002) is a very thorough and good introduction to mathematical logic in which all theorems are numbered; it also introduces to higher-order logic (see next chapter). Unfortunately, the book is rather expensive.

- H.-D. Ebbinghaus, J. Flum, and W. Thomas (1994). *Introduction to Mathematical Logic*. Springer.
- Andrews, Peter B. (2002). *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer.

Higher-Order Logic

In this chapter we will take a look at higher-order logic insofar as it is used in linguistic theorizing. The chapter provides more of an overview and does not go into all technical details. The version of higher-order logic we will be concerned with is also sometimes called *type theory*, because every expression in this logic must have a semantic type that regulates how it is interpreted. One of the most influential type theories was Church's *simple theory of types*, which will also be the basis of this chapter. Type theory is surprisingly simple and elegant; many concepts are definable in it because of its expressivity. This comes at a price: Higher-order logic with standard models is not compact and not complete. There are, however, proof theories that are complete for higher-order logic with so-called General Henkin Models and there are many automated theorem provers for higher-order logic.

4.1 Syntax of Simple Type Theory

We call our type theoretic language *HOL*. The formulation will differ from the seminal one introduced by Church (1940).

4.1.1 Types

Types. Let T be a finite set of types of *HOL*. For the time being, we only use e for entities and t for truth values. If $\alpha, \beta \in T$ then $(\alpha\beta) \in T$. Nothing else is in T .

Notice that the base types are the same as the ones we used for first-order logic, but compound types hint at some richer expressivity. An ordinary first-order unary predicate has type (et) ; in *HOL* there are infinitely many types on top of the base type. For example, $((et)t)$ is a type, which will later be interpreted as a predicate of a predicate.

Notational Convention. Outer parentheses around types may be left out. Within types parentheses may be left out, in case of which right-associativity is assumed. This means that for example eet may be written for $e(et)$ and $(et)(et)$ for $((et)(et))$.

4.1.2 Terms

Base Terms. Base terms consist of sequences of alphanumeric characters and special symbols like \forall, \exists, \wedge , and so forth. Let L_T be the set of terms of *HOL* containing the base terms. Every base term has a base type that may be indicated as a superscript.

Variables. We assume that for every term of type α there is a variable of type α , where by convention x, y, z are used as variables of type e and P, Q, R as variables of type et as long as no other type is indicated as a superscript.

Constants. $N^{tt}, A^{t(tt)},$ and $Q^{\alpha(at)}$ are logical constants, where α is any type. We use ϕ, ψ as metavariables for terms in what follows.

Compound Terms.

- i. If ϕ is a base term of type $(\beta\alpha)$ and ψ a term of type β , then $(\phi\psi)$ is a term of type α .
- ii. If ϕ is a term of type α and x is a variable of type β , then $(\lambda x\phi)$ is a term of type $(\beta\alpha)$.

✧ **Remark 9 (Alternative Notations.)** Using e as type for objects and t for truth-values is common in the linguistic literature. There is less agreement on so-called intensional types, i.e. types for terms that denote entities like possible worlds or situations. The letter i (for intension) is sometimes used; I personally have used s for situation and c for context elsewhere. Linguists often write types like the tuples by means of which they can be represented: $\langle\langle e, t \rangle t \rangle$ instead of $(et)t$. Computer scientists often compose types with \rightarrow as a symbol: $(e \rightarrow t) \rightarrow t$ instead of $(et)t$. (They also sometimes use additional type constructors like for example \times for so-called product types. For example the type $(e \times e)t$ would denote a binary predicate. We do not use these types here.) In the logical literature on simple type theory Church's notation seems to be prevalent. He uses ι for objects, o for truth-values, left-associativity is assumed, and the order is reversed in comparison to our types. That is, ooi in Church's notation corresponds to $(et)t$ in our notation.

☞ **Note 13 (Type Theory vs. Higher-Order Logic)** You might have realized that there are no wffs in type theory. This is so, because as we will see below the types suffice for understanding what a term means. Moreover, type theory alone doesn't require us to have a type t for truth-values. While our semantics in the next section is for higher-order logic, where there is at least a type e for individuals and a type t for truth-values, the λ -calculus that will briefly be strived in section 4.3 works independently of whether there is a type t or not. In this sense, higher-order logic is a special application of simple type theory. For completeness it must be mentioned, however, that the above way using types is just one popular among many possible ways of formulating a paradox-free higher-order logic. Some formulations of higher-order logic do not use types at all and instead avoid paradoxes by restricting a *comprehension principle* that regulates the concepts to which a higher-order variable may refer in a suitable way.

The reason why it is relatively common to use A , N , Q , and Π as symbols is that the functions are Schönfinkelized (see page 101 below) and some syntactic sugar needs to be added to make terms more readable.

☞ **Note 14 (Prefix vs. Operator-Argument Syntax)** Following a tradition based on Church's simple theory of types the above syntax specifies prefix notation for functions instead of the familiar functor-argument syntax. In prefix notation, the parentheses are put around the functor and its argument, i.e. $(f a)$ is written instead of $f(a)$ for the functional application of f to its argument a . This syntax has been popularized by the programming language *LISP* and its derivatives, which originally has been based on λ -calculus plus a few built-in functions.

Notational Conventions. We may write $\neg\phi$ for $(N\phi)$, $(\phi \vee \psi)$ for $((A\phi)\psi)$, and $(\phi = \psi)$ for $((Q\phi)\psi)$. Let us also allow functor-argument syntax as a notational variant and other usual syntactic conventions such as infix notation for $=$, dot-notation, and leaving out redundant parentheses. Let us further write functions as taking n argument instead of subsequent applications of unary functions. That is, we may write $P(x, y)$ instead of $((P x)y)$. Finally, it is common to contract multiple λ -abstractions into one, i.e. for example to write $\lambda xy.Pxy$ for $\lambda x\lambda y.Pxy$.

Because of the rules for the λ -calculus that will be introduced in section 4.3 great care must be taken not to confuse the scope of λ -terms and their arguments when the notations are mixed, though. In case of doubt, it is best to resort to the original preafix notation and I will use only few notational simplifications in what follows. You should also keep in the back of your mind that n -ary function notation is only syntactic sugar in traditional non-relational type theory. For example, when we write $P(a, b)$ as usual, where P is of type eet and a, b are of type e , it must be kept in mind that there is an intermediate term (Pa) whose meaning is well-defined: it is a function taking an object and yielding a function that takes an object and yields a truth value.

4.2 Semantics of Higher-Order Logic

4.2.1 General Models and Truth in a Model

General Models. A general model M of *HOL* contains a base domain D_α for each base type α and an interpretation function $I^g(\cdot)$ that maps expressions to their domain in dependence of a variable assignment g as follows:

$$I^g(\phi^\alpha) \in D_\alpha \quad (4.1)$$

$$I^g(\phi^{(\alpha\beta)}) \in D_{(\alpha\beta)}, \text{ where } D_{(\alpha\beta)} \subseteq D_\beta^{D_\alpha} \quad (4.2)$$

$$I^g(\lambda x^\alpha. \phi^\beta) \text{ is the function } f \text{ such that for any } \quad (4.3)$$

$$a \in D_\alpha, f(a) = I^{g[x/a]}(\phi)$$

Definition 4.2 is of particular importance. Notice that it defines the domain of terms of a function type $(\alpha\beta)$ as a *subset* of all functions from D_α to D_β . In practice, when a concrete model is specified, this means that we have to choose a *particular* subset of the set of all functions from D_α to D_β for each compound type $(\alpha\beta)$ in the model, and consequently the quantifiers for variables of type $(\alpha\beta)$ only run over this subset. If we had instead defined $D_{(\alpha\beta)} = D_\beta^{D_\alpha}$ then the model would be a so-called *standard model* and the logic as a whole would have very different properties from the one we have defined here! (See section 4.6 for more details.) General models go back to Henkin (1950) and are often called Henkin models.

Truth in a Model. To define the logical constants N , A , and Q , a function $\llbracket \cdot \rrbracket^{M,g}$ evaluates expressions in dependence of I of M and an assignment g as follows:

$$\llbracket (N\phi) \rrbracket^{M,g} = \begin{cases} 1 & \text{if } \llbracket \phi \rrbracket^{M,g} = 0 \\ 0 & \text{otherwise} \end{cases} \quad (4.4)$$

$$\llbracket A \rrbracket^{M,g} \text{ is the function } f \text{ such that } f(x)(y) = 0 \quad (4.5)$$

$$\text{if } x = 0 \text{ and } y = 0, \text{ and otherwise } f(x)(y) = 1$$

$$\begin{aligned} \llbracket Q \rrbracket^{M,g} \text{ is the function } f \text{ such that } f(x)(y) = 1 & \quad (4.6) \\ \text{if } x = y, \text{ and otherwise } f(x)(y) = 0 & \end{aligned}$$

$$\llbracket (\phi\psi) \rrbracket^{M,g} = I^g(\phi)(I^g(\psi)) \quad (4.7)$$

$$\llbracket (\lambda x\phi) \rrbracket^{M,g} = I^g(\lambda x\phi) \quad (4.8)$$

This is one way of defining a minimal higher-order logic. N is obviously just truth-functional negation, A is disjunction, and Q is identity. The other rules just pass on functional application and λ -abstraction to the previously defined interpretation function. The formulations of disjunction and identity might seem odd at first glance, and there would have been numerous other ways to define them, but they make sense given that *HOL* only contains unary functions. In the above formulations, the dyadic functions have been *Schönfinkelized* or, according to more common terminology, *curried*.

4.2.2 Interdefinability of Quantifiers and Identity

Where are the quantifiers? As it happens, identity and the universal or existential quantifiers are interdefinable in higher-order logic. Here is how to define the universal quantifier:

$$\top := ((Q^{(ttt)(ttt)t} Q^{(ttt)}) Q^{(ttt)}) \quad (4.9)$$

$$(\Pi(\lambda x\phi)) := ((\lambda x\phi) = (\lambda x.\top)) \quad (4.10)$$

The auxiliary definition (4.9) is a cumbersome, but correct way of defining the Verum, i.e. a term that is always true. Definition (4.10) then defines the universal quantifier Π by asserting that the term $\lambda x\phi$ is identical to a function that takes an x and yields true. Bear in mind that $\llbracket \lambda x^\alpha\phi \rrbracket^{M,g}$ is the function f such that for any $a \in D_\alpha$, $\llbracket \phi \rrbracket^{M,g[x/a]} = 1$. The identity in definition (4.10) asserts that this function f yields true for any given a , i.e. for all $a \in D_\alpha$, $\llbracket \phi \rrbracket^{M,g[x/a]} = 1$. That's universal quantification. Since the symbol Π looks a bit weird it is customary to write $\forall x\phi$ instead of $\Pi(\lambda x\phi)$. The existential quantifier can be defined as $\exists x\phi := \neg\Pi(\lambda x\neg\phi)$.

You already know from the discussion of Leibniz' Law in the last chapter how to define identity in terms of the universal quantifier. Instead of (4.6) we could have introduced a Schönfinkelized definition for Π , defined \forall as a notational shortcut and then used formula 3.15 on page 73 to introduce identity.

Notational Convention. We may write $\forall x\phi$ instead of $\Pi(\lambda x.\phi)$.

4.2.3 More Definitions

The other truth-functional connectives are defined as usual. For completeness, here are some definitions you could use:

$$(\phi \wedge \psi) := \neg(\neg\phi \vee \neg\psi) \quad (4.11)$$

$$(\phi \rightarrow \psi) := (\neg\phi \vee \psi) \quad (4.12)$$

$$(\phi \leftrightarrow \psi) := ((\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)) \quad (4.13)$$

4.3 Typed λ -Calculus

There are tableaux systems for higher-order logics and they look very similar to the ones for first-order logic plus some rules for λ -abstractions. Proof systems like tableaux or natural deduction rules for higher-order logics are primarily used (in combination with more specialized implementation methods) in higher-order logic theorem provers – see, for example, the following ones:

- Isabelle <http://www.cl.cam.ac.uk/research/hvg/isabelle/>
- Leo-II <http://www.ags.uni-sb.de/~leo/>
- TPS <http://gtps.math.cmu.edu/tps.html>¹

Using a proof theory for higher-order logic by hand is cumbersome and we will not take a look at tableaux for higher-order logic here. However, using the rules of *typed λ -calculus* is very common in semantics. Lambda calculus only deals with λ -terms, which according to rule 4.3 express functional abstractions. As we shall see soon, they can be used to express arbitrary scope distinctions and for this reason play a crucial role in semantics. The typed λ -calculus allows us to simplify λ -terms in a mechanical way and thereby resolve scope distinctions.

4.3.1 Conversion Rules

The rules of lambda calculus are known as α -, β -, and η -conversion. Here they are:

$$(\lambda x.\phi) \stackrel{\alpha}{\Leftrightarrow} (\lambda y.\phi[x/y]) \quad \alpha\text{-conversion} \quad (4.14)$$

$$((\lambda x.\phi)\psi) \stackrel{\beta}{\Leftrightarrow} \phi[x/\psi] \quad \beta\text{-conversion} \quad (4.15)$$

$$(\lambda x.\phi) \stackrel{\eta}{\Leftrightarrow} \phi \quad \eta\text{-conversion} \quad (4.16)$$

Hereby, x may not be free in ϕ in 4.16, x must be free in ϕ and x and ψ must be of the same type in 4.15, and $\phi[x/y]$ is the same term as ϕ except that all free occurrences of x in ϕ have been substituted by y .

The rules express equivalences, but the \Leftrightarrow sign must be understood as a purely syntactic operation. In other words, the rules really specify a calculus, i.e. a mechanical system for calculating something that is like a proof theory but

¹ If it weren't against other regulations I wouldn't mind if you would use one of these theorem provers during exams. Learning and understanding how to use them is not substantially easier than learning and using tableaux. The educational variant of TPS called ETPS is used at Carnegie Mellon university for introductory logic courses.

more general (every proof theory is a calculus but not vice versa). The expression on the left hand side may be rewritten as an expression on the right hand side and vice versa. Since in rules 4.15 and 4.16 the expressions on the right hand side are less complex than the ones on the left hand side, they are usually used in the direction from the left to the right in order to simplify terms. Used from left to right, we call the rewrite rules *reduction rules*. Thus, it is common to speak of β -reduction or η -reduction.

4.3.2 λ -Abstraction at Work

The rule for β -reduction is used very often. It says that you may substitute a constant of the same type as the variable x in a term $\lambda x.\phi$ for free occurrences of that variable in ϕ . To see why this is relevant to linguistic theorizing, consider the following example.

✳ **Example 9 (β -Reduction.)** We would like to specify the truth-conditional content of the following sentence (ignoring tense and aspect):

(4.17) O João gosta da Maria.

Let ‘*Maria*’ and ‘*João*’ be terms of type e . Let ‘*like*’ be a term of type eet . Then:

$$(\lambda y \lambda x ((\textit{like } x) y) \textit{Maria}) \textit{João} \quad (4.18)$$

$$\xrightarrow{\beta} \lambda x ((\textit{like } x) \textit{Maria}) \textit{João} \quad (4.19)$$

$$\xrightarrow{\beta} (\textit{like } \textit{João}) \textit{Maria} \quad (4.20)$$

As you can see the order of argument application can be easily reversed by using λ -abstraction. Of course, in the above example this is not really necessary, because the function ‘*like*’ could simply have been defined with the arguments reversed, i.e. reading *like* $x y$ iff y likes x instead of the ‘more natural’ order of arguments. However, in many other cases λ -abstraction is needed. The following example further illustrates the power of λ -abstraction.

✳ **Example 10 (VP-Conjunction.)** We would like to specify a semantic representation for the following sentences (ignoring tense and aspect):

(4.21) O João ama e odia a Maria.

Let ‘*hate*’ be a function of type eet and ‘*and*’ be of type $(eet)((eet)(eet))$ defined as $\lambda P^{eet} \lambda Q^{eet} \lambda y \lambda x. ((P x) y) \wedge ((Q x) y)$.

$$(((\textit{and love}) \textit{hate}) \textit{Maria}) \textit{João} \quad (4.22)$$

$$= (((\lambda P^{eet} \lambda Q^{eet} \lambda y \lambda x. ((P x) y) \wedge ((Q x) y)) \textit{love}) \textit{hate}) \textit{Maria}) \textit{João} \quad (4.23)$$

$$\xrightarrow{\beta} (((\lambda Q^{eet} \lambda y \lambda x. ((\textit{love } x) y) \wedge ((Q x) y)) \textit{hate}) \textit{Maria}) \textit{João} \quad (4.24)$$

$$\stackrel{\beta}{\Rightarrow} ((\lambda y \lambda x.((\textit{love } x) y) \wedge ((\textit{hate } x) y)) \textit{Maria}) \textit{João} \quad (4.25)$$

$$\stackrel{\beta}{\Rightarrow} (\lambda x.((\textit{love } x) \textit{Maria}) \wedge ((\textit{hate } x) \textit{Maria})) \textit{João} \quad (4.26)$$

$$\stackrel{\beta}{\Rightarrow} ((\textit{love } \textit{João}) \textit{Maria}) \wedge ((\textit{hate } \textit{João}) \textit{Maria}) \quad (4.27)$$

As you can see, we're getting closer and closer to natural languages. We have just learned about a way to specify a reasonable semantic representation for any phrase of the form *VP and VP*, where the *VPs* are transitive verbs. Before exploring the possibilities of representing the truth-conditional semantics of natural language expressions in higher-order logic in more detail, an important remark has to be made about the examples. It is absolutely crucial to be aware of the fact that the names of our object-language functions such as '*Maria*', '*like*', or '*and*' play no role in actual linguistic theorizing. Just like in the previous chapter, we could have used letters like *P* or symbols like \wedge instead. The semantic type of these functions counts, as would special any meaning rules we might specify for them, and because of the fixed meaning of \wedge and functional application the result 4.27 expresses some structural constraint – but the names of the functions are insignificant.

The α -conversion rule is also important, because of the restriction for β -reduction that x must be free in ϕ . Suppose we have a term $\lambda P^{et}.\exists x[Px]$, where x is bound by the existential quantifier. Suppose we want to apply this term to $\lambda x.x = x$; we are not allowed to apply the β -reduction rule, because x is not free in $\exists x[Px]$. But we are allowed to exchange variables in either of the terms using α -conversion, e.g. we may convert $\lambda x.x = x$ to $\lambda y.y = y$ and then apply β -reduction as follows:

$$(\lambda P^{et}(\exists x[Px])) (\lambda x.x = x) \quad (4.28)$$


$$\stackrel{\alpha}{\Rightarrow} (\lambda P^{et}(\exists x[Px])) (\lambda y.y = y) \quad (4.29)$$

$$\stackrel{\beta}{\Rightarrow} \exists x[(\lambda y(y = y)) x] \quad (4.30)$$

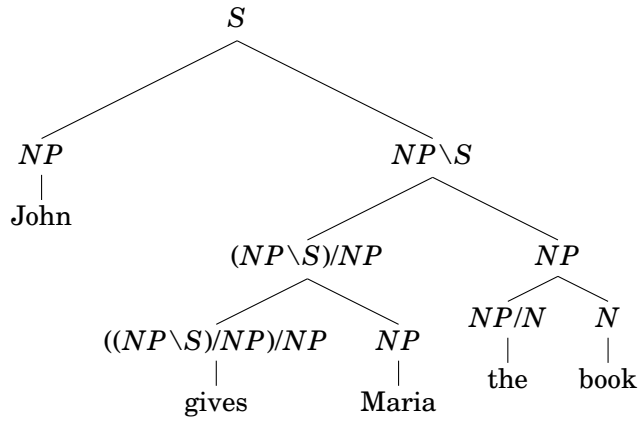
$$\stackrel{\beta}{\Rightarrow} \exists x[x = x] \quad (4.31)$$

Finally, the purpose of the η -conversion rule is easy to see. It allows us to get rid of vacuous λ -abstraction. Take a term like $\lambda x.\textit{like}$. The x is never applied to the function and so the abstraction is vacuous, or *spurious* as it is sometimes also called. Using η -reduction, we can rewrite this term as '*like*'.

4.3.3 Exercises

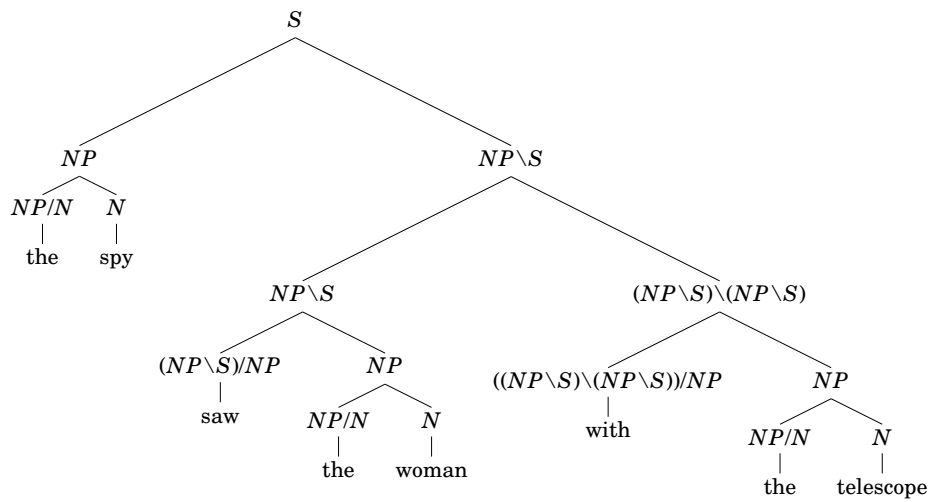
 **Exercise 39** Simplify the following terms in preafix notation using rules 4.14 to 4.16:

b. John gives Maria the book.

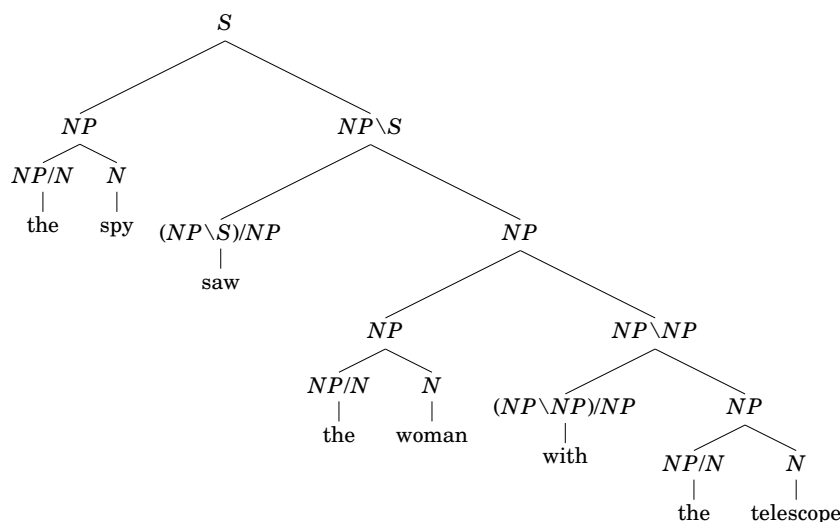


c. The spy saw the woman with the telescope.

d.



e.



Notice that ‘with’ has been given two different syntactic categories in order to account for the two different readings. With category $((NP \setminus N) \setminus (NP \setminus N)) / NP$ it takes an NP and then a verb phrase to combine with the verb phrase. When it has category $(NP \setminus NP) / NP$ it combines with two NPs and yields a new NP , giving the other reading. For the semantics in the next section this scheme will have to be slightly modified, but the general idea will remain the same.

4.4.2 Type-driven Evaluation

We now add the semantic component to our categorial grammar. To do this, we interpret the lexical items directly as terms of higher-order logic and make sure that to any compound syntactic category a/b and $b \setminus a$ belongs a corresponding semantic type (ab) . As a result, it is possible to derive the semantics of an expression directly in parallel to its syntax by interpreting the syntactic operations of forward and backward concatenation as functional application. Consider sequences of terms. These are interpreted according to the following phrase structure rule:

$$(a/b) : A^{(\alpha\beta)} b : B^\beta \xrightarrow{f} a : (AB) \quad (4.32)$$

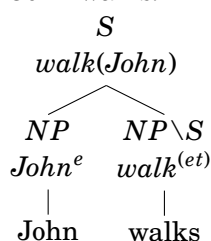
$$b : B^\beta (b \setminus a) : A^{(\alpha\beta)} \xrightarrow{b} a : (AB) \quad (4.33)$$

The rules are to be read as follows: To evaluate a sequence of two terms AB of the respective syntactic categories and semantic type, rewrite them as on the right-hand side, where the new syntactic category is indicated. This process is known as type-driven evaluation. For this to work we must assure a close correspondence between syntactic categories and semantic types which may be called a category-type well-formedness principle. Only sequences of terms with categories and types like in the above schemes are well-formed. This way,

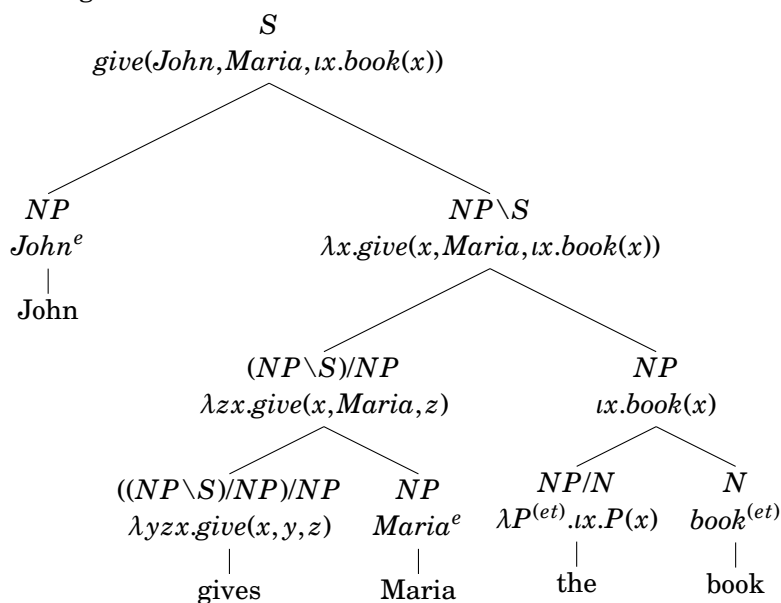
syntax and semantics are kept in parallel. To see how this works, take a look at the examples from the previous section where now the semantics is annotated below the syntactic categories.⁴

✱ **Example 12 (Categorial Grammar)** Let S, NP, B be our syntactic base categories. Rule applications can be depicted by trees as follows. Hereby, tense, mood, and aspects are ignored and I use traditional operator-argument syntax for applications of functions like *walk* or *give* that are not further analyzed. Moreover, the semantic type of terms is only annotated once and not repeated.

a. John walks.

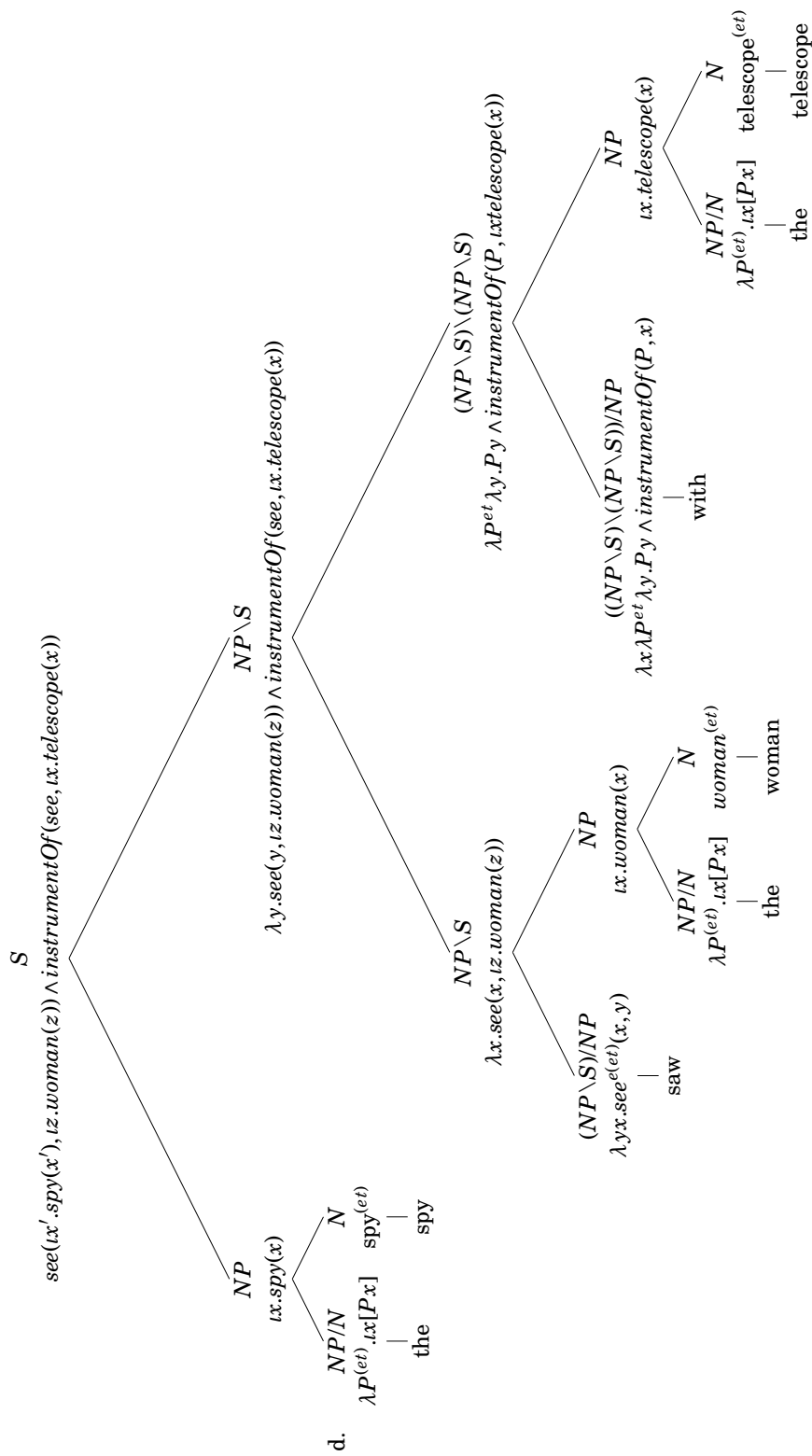


b. John gives Maria the book.

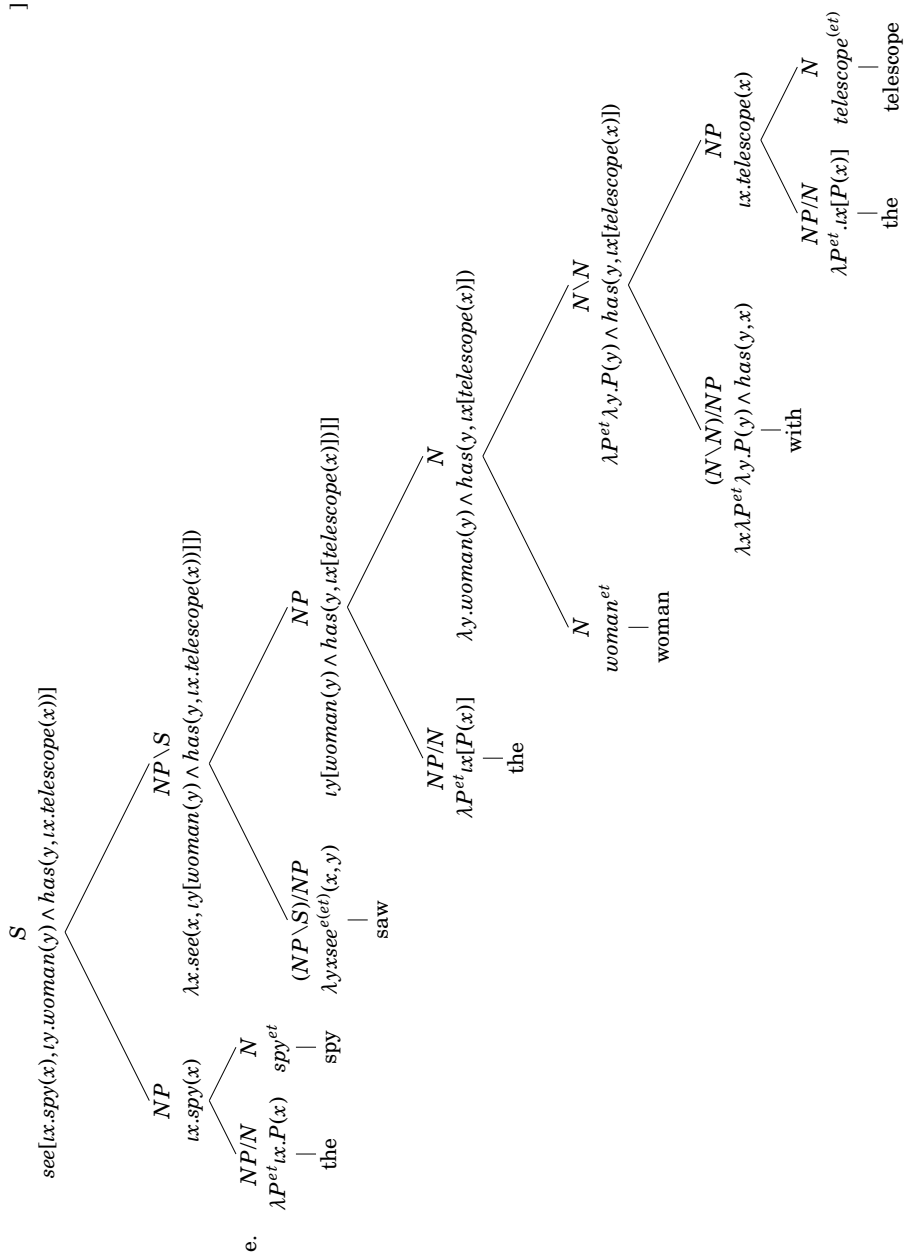


c. The spy saw the woman with the telescope.

⁴ For better readability I put the actual lexical strings on nodes of their own as if there were lexical insertion rules like in phrase structure grammar.



]]



Actually, example c is not quite correct but at least close to what would be desirable. The reason why the semantic representation is strictly speaking not correct is that according to the end result *instrumentOf* modifies *see* in general. In reality, however, the telescope is only the instrument for watching the woman in *this particular* situation at the given time described by the sentence. For a good solution to this problem we need to introduce situations or events.

4.5 Applications

4.5.1 Verbs, Proper Names

Strictly speaking, in the present setting n -ary relations cannot be expressed and need to be encoded by Schönfinkelizing them into multiple functions of one argument. As mentioned earlier, we can, however, write $P(x, y)$ for $((P x) y)$. For all practical purposes we can use the logic as if it had relations directly in the object language. That being said, the translations of verbs are like in the previous chapter. Not taking into account tense, aspect, or sentence mood verbs can be expressed as follows:

- an intransitive verb is represented by a unary predicate,
- a transitive verb is represented by a binary relation,
- a ditransitive verb is represented by a ternary relation, and
- in general a verb that requires n mandatory arguments is represented by an n -ary relation.

The maximum number of obligatory arguments is not very high in natural languages, 4-5 seems to be the maximum; these numbers depend on the criteria chosen to determine when an argument is ‘obligatory’. You should consult literature in *lexical semantics* for more (authoritative) information.

Once we also take into account possible worlds, situations, events, or contexts the argument place of predicates respectively change. One or even two argument places might be added to each predicate in such an intensional framework. However, even in these frameworks adding additional argument places is not always necessary and theory-dependent.

4.5.2 Generalized Quantifiers

Recall generalized quantifiers introduced in the first chapter. For example, the truth conditions for the quantifying determiner ‘some’ could be expressed in set theory in terms of conditions between a set for the quantifier restriction and the quantifier body. ‘Some teachers are lazy’ can be expressed in terms of a condition between the set of teachers and the set of lazy ‘objects’:

$$\{x \mid x \text{ is a teacher}\} \cap \{x \mid x \text{ is lazy}\} \neq \emptyset$$

We can now express generalized quantifiers directly in the object language, meaning that we can directly derive the appropriate semantic representation from the syntax and the lexicon. However, in the above examples the syntactic category of an intransitive verb phrase like ‘giggles’ or, for what its worth, ‘are lazy’ (simplified, of course) is $NP \setminus S$ and the corresponding semantic type was et , i.e. a function taking an individual and yielding a truth value. These types don’t work for generalized quantifiers, because for example the quantifier ‘all students’ represents a set of students and not just one.

A solution is to ‘shift up’ the type of the generalized quantifier to actually take the meaning of the verb phrase (instead of vice versa) and yield a sentence-type meaning with the correct truth-conditions. Since the type of the verb phrase is et , the type of the generalized quantifier must be $(et)t$, i.e. a function that takes a unary predicate of type et and yields a truth-value. Correspondingly, the syntactic category of a generalized quantifier must be $S/(NP \setminus S)$. What about quantifying determiners themselves then, i.e. expressions like ‘some’, ‘most’, ‘a’, ‘no’, or ‘all’?

Since the type of a countable noun like ‘cat’, ‘dog’, ‘teacher’, or ‘student’ is also et , a unary predicate expressing a property or a set of objects in the extensional view, and its syntactic category in the present setting is N , the type of a generalized determiner must be $(et)((et)t)$, i.e. a function that takes a unary predicate (the meaning of the noun) and yields a function that takes another unary predicate (the meaning of the intransitive verb) and yields a truth value as a meaning of the whole sentence. Correspondingly, the syntactic category of a generalized determiner in languages like English or Portuguese must be $(S/(NP \setminus S))/N$: it takes a noun from the right and yields an expression that consumes a verb phrase from the right to yield a sentence. Here are example entries:

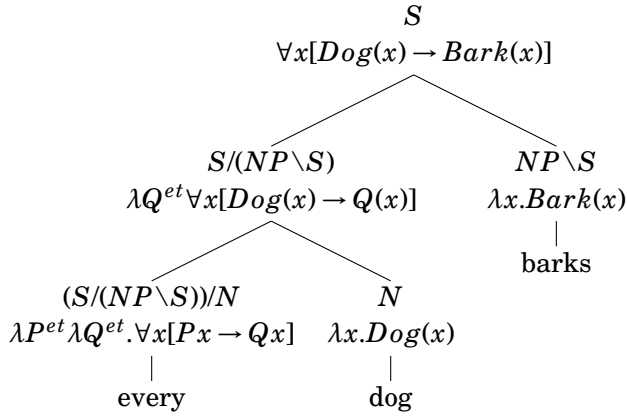
$$every := (S/(NP \setminus S))/N : \lambda P^{et} \lambda Q^{et} . \forall x [P x \rightarrow Q x] \quad (4.34)$$

$$some := (S/(NP \setminus S))/N : \lambda P^{et} \lambda Q^{et} . \exists x [P x \wedge Q x] \quad (4.35)$$

$$no := (S/(NP \setminus S))/N : \lambda P^{et} \lambda Q^{et} . \neg \exists [P x \wedge Q x] \quad (4.36)$$

$$a := \text{like } some \quad (4.37)$$

* **Example 13** Every dog barks.



4.5.3 Generalized Quantifiers and the Finite Verb Phrase

A transitive verb can be handled in the same manner as an intransitive verb, except that it first needs to combine with the direct object. This, however, means that unless we provide for additional mechanisms we need to type-shift the type of generalized quantifiers to account for their occurrence in the position of the direct object. Assuming that the type of individuals e is the one we base our entries for verbs on, the type of a transitive verb phrase must be $e(et)$ and the corresponding syntactic category is $(NP \setminus S)/NP$.

We get the lexicon entries that are even further ‘shifted up.’ The generalized quantifier in direct-object position takes transitive verb and applies the meaning of the direct object NP to it. The result is an entry of category $NP \setminus S$ and type et :

$$every := (((NP \setminus S)/NP) \setminus (NP \setminus S))/N : \lambda P^{et} \lambda Q^{e(et)} \lambda y. \forall x[Px \rightarrow Qyx] \quad (4.38)$$

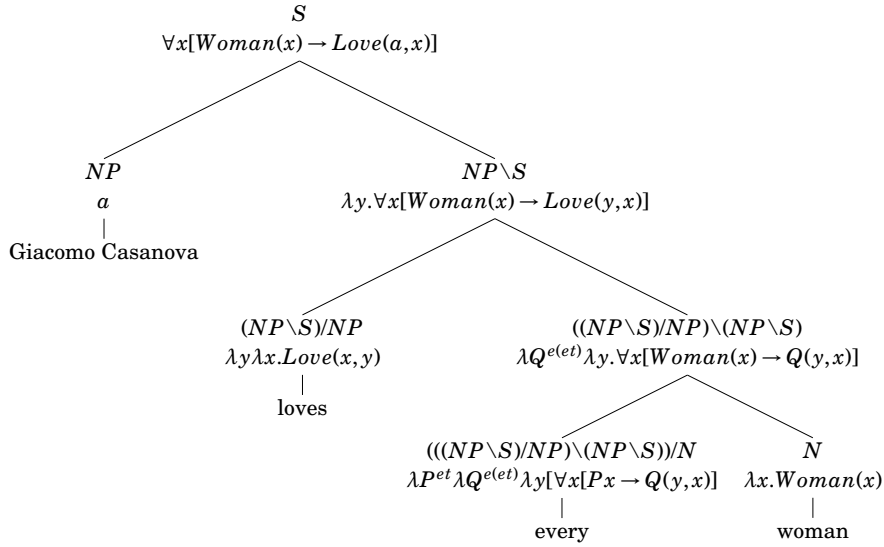
$$some := (((NP \setminus S)/NP) \setminus (NP \setminus S))/N : \lambda P^{et} \lambda Q^{e(et)} \lambda y. \exists x[Px \wedge Qyx] \quad (4.39)$$

$$no := (((NP \setminus S)/NP) \setminus (NP \setminus S))/N : \lambda P^{et} \lambda Q^{e(et)} \lambda y. \neg \exists [Px \wedge Qyx] \quad (4.40)$$

$$a := \text{like } some \quad (4.41)$$

To see how this works, let us take a look at an example in which only the direct object is semantically represented by a generalized quantifier.

* **Example 14** Giacomo Casanova loves every woman.



4.5.4 Quantifier Scope Ambiguities

Ditransitive verbs like in ‘Many teachers give the book to every student’ require quantifiers and quantifying determiners to be shifted up even higher.⁵ General type shifting principles have been stipulated that yield the desired lexicon entries in a systematic way. However, there is another problem with the approach caused by quantifier scope ambiguities. Consider the following sentence:

(4.42) Every sailor loves a woman.

In the first and prevalent reading, every sailor loves some woman (in a contextually restricted domain) but not necessarily the same one. In a second reading, there is one woman, say Rosy, whom every sailor (in the contextually restricted domain) loves. The above type-shifted lexicon entries only account for the first reading:

$$\forall x[Sailor(x) \rightarrow \exists y(Woman(y) \wedge Love(x,y))] \quad (4.43)$$

To derive the second reading, \exists needs to have scope over \forall :

$$\exists x[Woman(x) \wedge \forall y(Sailor(y) \rightarrow Love(y,x))] \quad (4.44)$$

It is possible to obtain this reading using only the mechanisms introduced so far by giving the quantifying indefinite article ‘a’ in direct-object position a syntactic category that basically consumes the whole rest of the sentence from the left.

⁵ Also notice the peculiar use of the definite determiner ‘the’ in this example, which cannot be represented adequately in the Russellian way as a iota term or quantifier.

The resulting syntactic and semantic composition is monstrous, though. Since this kind of shifting is ad hoc and undesirable from a formal point of view many other solutions to quantifier scope ambiguities have been explored:

- In the transformational and generative grammar tradition, the categorial grammar is only used on the semantic side to approximate the semantic representation to the actual syntax, which is based on the constituent structure of the sentence. Readings like (4.44) can be obtained by transformation rules when deriving the logical form from an underlying syntactic representation while keeping the amount of type shifting as minimal as possible.
- In computational linguistics, algorithms have been developed that can be used to derive all possible readings triggered by quantifier scope ambiguities – and not all readings one might naively think are possible are in fact possible. These algorithms and their corresponding stack-based data structures are known as *Cooper storage* and *Keller storage*.
- In the tradition of Type Logical Grammar, the above applicative categorial grammar is only considered a fragment of the more expressive full-fledged categorial grammar based on so-called Lambek calculus, which also allows for assigning meanings to non-constituent expressions like ‘Every sailor likes.’ Moreover, many more ways of combining meanings are available that allow for resolving quantifier scope ambiguities more elegantly.

4.5.5 Outlook and Limits

Notice that problems like quantifier scope ambiguity concern the syntax–semantics interface and solutions to these kinds of problems generally depend on the underlying syntactic theory. Semantic construction can look quite differently from the perspective of theories like Chomsky’s Minimalism, Lexical Functional Grammar, Tree Adjoining Grammar, Type-Logical Grammar and Combinatory Categorical Grammar, or HPSG.

It is, however, reasonable to believe that practically all *semantic phenomena* can be represented adequately in higher-order logics, simply because these logics are so expressive. In particular, all sorts of modification can be expressed adequately. It was for example mentioned in the last chapter that ‘famous’ in ‘famous pianist’ is not an intersective adjective. In a higher-order logic, ‘famous’ can be elegantly represented as a function $(et)(et)$ from a function of type et to a function of type et . Intensifiers like ‘very’ or ‘pretty’ as in ‘it’s very hot in here’ can be analyzed by the same token: For example, ‘very’ can be considered a function of type $((et)(et))(et)(et)$, i.e. a function that takes an adjective and yields an (intensified) adjective. While in many cases the same result can be achieved with some trickery (keyword: reification) in first-order predicate logic, the analysis in simple type theory is usually more natural and elegant.

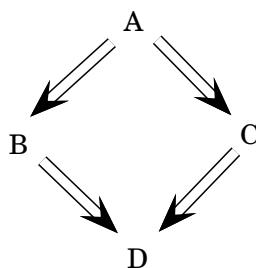


Figure 4.1: The Church-Rosser property.

4.6 Metatheorems

Higher-order logic differs in expressivity depending on the models that are allowed. Generally speaking, very powerful concepts can be expressed in higher-order logic and models can be chosen accordingly. For example, one might pick models in which the axiom of choice of set-theory is true or allow only those in which it turns out false. Or one might allow only those models in which the continuum hypothesis is true (or false, respectively). As one might imagine, even seemingly small changes like that of going from standard to Henkin models can have huge consequences, and a large part of research on higher-order logic is intertwined with research on the foundations of mathematics. Proofs of metatheorems are generally harder than for first-order logic and the properties of a particular formulation of higher-order logic and models for it depend very much on the detail.

Church-Rosser Theorem. According to the Church-Rosser theorem,⁶ when the evaluation of a term A of the λ -calculus splits up into two paths then the two resulting terms B and C will always be reducible to the same third term D . This is also called the diamond property and illustrated by figure 4.1.

When a rewrite system has this property it is also said to be *confluent*: The rules of the rewrite system guarantee that when there are two ways to proceed with the rewriting (for example by order of rule application) then the two rewriting ‘paths’ will eventually flow together.

Normalization Theorems. According to the weak normalization theorem every term of simply typed λ -calculus can be brought into a normal form. According to the strong normalization theorem no term of simply typed λ -calculus has an infinite reduction sequence, i.e. no term requires infinitely many reduction

⁶ See Alonzo Church and J. Barkley Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, vol. 39, No. 3. (May 1936), 472–482.

steps in order to bring it into a normal form.

Taken together these theorems ensure that every λ -term can be brought into a normal form in finitely many steps and the reduction does not suddenly go astray or continue ad infinitum. It also means that there is an equivalence between terms modulo variable substitution by α -conversion that can be used to check syntactically whether two terms are equal.

Completeness and Incompleteness Results. Higher-order logic with standard models is incomplete. There are complete proof theories for versions of higher-order logic based on simple type theory with General models (Henkin 1950).

Compactness and Lack of Compactness. Higher-order logic with standard models is not compact. There are versions of higher-order logic based on simple type theory with Generalized Henkin models that are weakly compact, where ‘weakly compact’ means compactness with respect to general models as opposed to standard models (see Andrews 2002: Ch. 5).

Literature

The technical intricacies of higher-order logic are laid out in the following seminal works:

- Church, Alonzo (1940). A Formulation of the Simple Theory of Types. *The Journal of Symbolic Logic*, 5, 56-68. Reprinted in Benzmüller et. al. (2008), 35-47.
- Henkin, Leon (1950): Completeness in the Theory of Types. *The Journal of Symbolic Logic*, 15, 81-91. Reprinted in Benzmüller et. al. (2008), 49-59.
- Benzmüller, C.; Brown, C. E.; Siekmann, J. & Statman, R. (eds.) (2008). General Models and Choice in Type *Theory Reasoning in Simple Type Theory*. College Publications.
- Andrews, Peter B. (2002). *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Kluwer.

None of them is particularly easy to read. As a start, I would recommend the book by Andrews to anyone who is seriously interested in mathematical logic and type theory. Benzmüller et. al. (2008) contains reprints of the most important articles on type theory.

The following works are first and foremost mentioned for historical interest. They are targeted at mathematicians and not suitable for beginners:

- Schönfinkel, Moses (1924). Über die Bausteine der mathematischen Logik. *Mathematische Annalen* 92, 305–316. Translation: On the building blocks of mathematical logic. In Jean van Heijenoort (1967). *A Source Book in Mathematical Logic*. Harvard University Press, 355–66.
- Haskell Curry and Robert Feys (1958). *Combinatory Logic I*. North Holland.
- Haskell Curry, J.R. Hindley and J.P Seldin (1972). *Combinatory Logic II*. North-Holland.

Moses Schönfinkel was a Russian mathematician who is known for his work on combinatoric logic. According to Wikipedia, ‘His later life was spent in poverty, and he died in Moscow some time in 1942. His papers were burned by his neighbors for heating.’ (English Wikipedia entry for ‘Moses Schönfinkel’ of 2010-08-27) Curry also worked on combinatory logic and is considered one of the most important contributors to the foundations of functional programming.

- Jon Barwise and Robin Cooper (1981). Generalized quantifiers and natural language. *Linguistics and Philosophy* 4, 159-219.
- Andrzej Mostowski (1957). On a generalization of quantifiers. *Fund. Math. Vol. 44*, 12-36.
- Barbara H. Partee, Alice ter Meulen, and Robert E. Wall (1990). *Mathematical Methods in Linguistics*. Springer.
- L.T.F. Gamut (1991). *Logic, language, and meaning*. Univ. of Chicago Press.
- Irene Heim and Angelika Kratzer (1998). *Semantics in a Generative Grammar*. Blackwell.

Generalized quantifiers go back to Mostowski (1957), have been used by Montague (1974) in *The Proper Treatment of Quantification in English* (PTQ), and have been investigated systematically by Barwise and Cooper (1981). Introductions can be found in ascending order of difficulty and detail in Heim and Kratzer (1981), Partee et. al. (1990), and Gamut (1991). By the way, the name ‘L.T.F. Gamut’ is a pseudonym for the collective of authors Johan van Benthem, Jeroen Groenendijk, Dick de Jongh, Martin Stokhof and Henk Verkuyl – an impressive collection of famous logicians.

- Richmond Thomason (ed.) (1974). *Formal Philosophy*. Yale University Press.
- David R. Dowty, Robert E. Wall and Stanley Peters (1981). *Introduction to Montague Semantics*. Kluwer.

The influence of Richard Montague's work on natural language semantics cannot be underestimated.⁷ The papers that are most important for linguists can be found in Thomason (1974). They are short, but very dense and presume a high level of technical expertise. For this reason Montague's work was mostly spread by some of his scholars such as Richmond Thomason, Barbara Partee, and David Dowty. Dowty et. al. (1981) is still the standard introduction to Montague Semantics and a good place to start.

Here is some of the seminal literature on type-logical grammar and combinatory categorial grammar mentioned above.

- Bob Carpenter (1997). *Type-Logical Semantics*. MIT Press.
- Michael Moortgat (1997). *Categorial Type Logics*. In Johann van Benthem and Alice ter Meulen (eds.). *Handbook of Logic and Language*. MIT Press, 93-178.
- Glynn Morrill (1994). *Type Logical Grammar: Categorial Logic of signs*. Kluwer Academic Publishers.
- Mark Steedman (1996). *Surface Structure and Interpretation*. MIT Press.
- Mark Steedman (2000). *The Syntactic Process*. MIT Press.

Carpenter (1997) is a very good introduction to type-logical grammar. It covers everything that has been covered in this chapter in detail in the first few chapters, introduces Lambek calculus using a sequent calculus and a natural deduction system, and then discusses the semantic modeling of a vast range of linguistic phenomena. Moortgat is a survey handbook article on categorial grammars in general and as such a good and formally rigid reference but not suitable for beginners. Morrill (1994) is a bit older but definitely worth reading. He starts from Montague's logic IL and quickly proceeds to more advanced topics; the book is very dense. (Morrill (2010) is not yet available at the time of this writing.) The books by Steedman are easy to read and intended for both beginning and experienced linguists interested in Combinatory Categorial Grammar. Steedman (2000) is more detailed and a good starting point.

⁷ He was murdered in 1971; the killer has never been found.



Solutions to Exercises

Chapter 1

Exercise 1, page 9:

- a. $\{1, 2, 3, 4, 5, 6\}$
- b. $\{(1, 1), \{1, 2\}, \{1, 3\}, \{1, 4\}, \{1, 6\}, \{2, 2\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{2, 6\}, \{3, 3\}, \{3, 4\}, \{3, 5\}, \{3, 6\}, \{4, 4\}, \{4, 5\}, \{4, 6\}, \{5, 5\}, \{5, 6\}, \{6, 6\}\}$

Note: Apart from these 20 unordered outcomes there are also 36 possible *ordered* outcomes of a throw of two standard dice. The case with 36 ordered pairs as an outcome is relevant for the calculation of the probability of an outcome.

Exercise 2, page 10:

- a. $\{n \in \mathbb{N} \mid \text{there is a } k \in \mathbb{N} \text{ s.t. such that } n = 2k + 1\}$
or, using the remainder function `mod` frequently available in programming languages:

$$\{n \in \mathbb{N} \mid n \bmod 2 = 1 \text{ and } n > 3\}$$

- b. $\{X \mid X \subseteq S\}$
Or simply: $\mathcal{P}(S)$
- c. $\{x \mid x \text{ is a blue sports car in Lisbon on October 24, 2014}\}$
Note: Using the indexical ‘today’ is not precise enough in a definition. Depending on the application, you would probably build this set out of sets representing the parts, e.g. the union of the set of objects in Lisbon on a certain day with the set of blue objects and with the set of sports cars.

- d. $\{X \mid X \subseteq A \text{ and there is a } Y \subseteq A \text{ such that } (X \cap Y) \neq \emptyset\}$

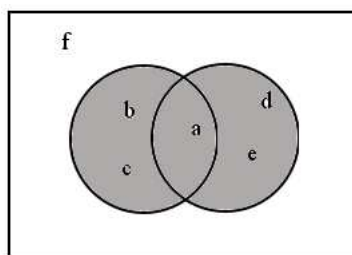
Exercise 3, page 10:

- a. $(A \cup B) \cap C = \{1\}$
 b. $(A \cap B) \cup C = \{1, 3, 4, 5, 9\}$
 c. $(A \setminus C) \cap B = \{3, 4, 5\}$
 d. $(C \setminus A) \cup ((B \cap A) \cup \emptyset) = \{9, 3, 4, 5\}$
 Note: $A \cup \emptyset = A$ for any A

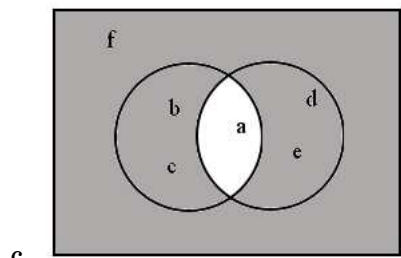
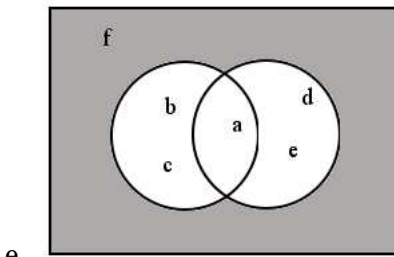
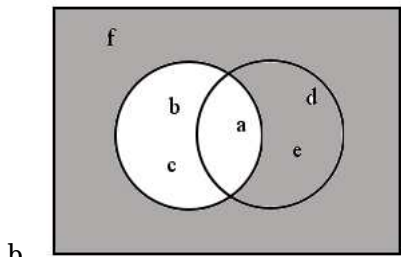
Exercise 4, page 11:

- a. It doesn't make sense, because sets are not ordered.
 b. Depending on the application it makes sense. Suppose you want to model an inventory of fruits.
 c. Let A be the set of employees, B be the set of union members, and C be the set of persons that get a higher salary. Then: $(A \cap \overline{B}) \subseteq C$
 d. Let A be the set of students and B be the set of workers.
 'Há estudantes que trabalham': $(A \cap B) \neq \emptyset$
 'Há estudantes que não trabalham': $(A \cap \overline{B}) \neq \emptyset$
 e. Let A be the set of students and B be the set of workers.
 'Todos estudantes trabalham ou não trabalham': $A \subseteq (B \cup \overline{B})$
 f. It is always true under the given analysis, because $A \subseteq (B \cup \overline{B})$ is always true. Notice, however, that for the presuppositional reading of the quantifier 'todos' we could add the restriction that $A \neq \emptyset$; under that presuppositional reading the sentence would be false if $A = \emptyset$.
 g. $(A \cap B) \cap C \neq \emptyset$
 Note: The order does not matter, we could have written $A \cap (B \cap C) \neq \emptyset$ and in fact the parentheses could be left out in this case.
 h. Yes, because the empty set is a subset of any set.

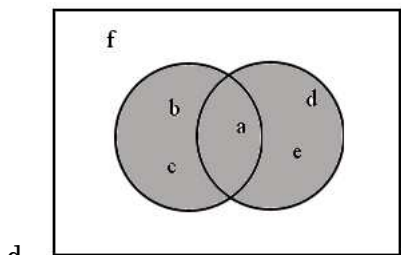
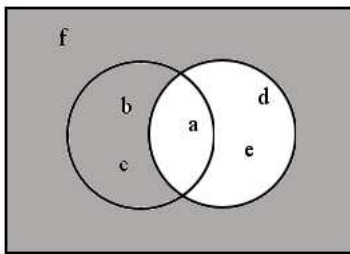
Exercise 5 on page 12:



a.

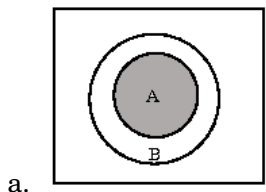


f. This does not hold, since $a \in A$ and $a \in B$. It is easy to see this from the diagram for \overline{B} :

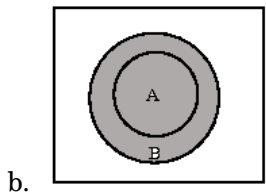


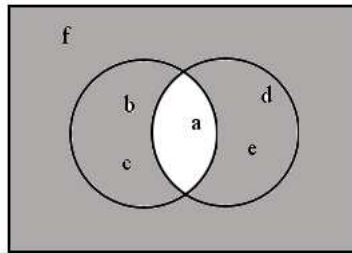
g. This does not hold, since $A \neq B$.

Exercise 6, page 12:



Note: This does not hold in general. It only holds when 1.) $A \subseteq B$, because then $A \cap B = A$ and thus $A \subseteq A$, or when 2.) $A = \emptyset$. The Venn diagram depicts the first case.





c.

Note: The grey area depicts $\overline{A \cap B}$. It is apparent from the picture that everything that is not in the grey area, i.e. the 'complement' of the grey area, is exactly $A \cap B$.

Exercise 7, page 13:

- a. $\{\emptyset, \{1\}, \{2\}, \{2, 1\}\}$
- b. $\{\emptyset\}$
- c. $\{\emptyset, \{c\}, \{a\}, \{a, c\}\}$

Exercise 8, page 13:

- a. $|A \cap B| \geq 5$
- b. $|A \cap B| = 1$
- c. $|A \cap B| \leq 3$
- d. 'not one' has (at least) two readings:
 - i. $A \cap B = \emptyset$ (read as 'no...')
 - ii. $|A \cap B| \neq 1$ (read as 'it is not the case that exactly one...')

Exercise 9, page 19:

- a. Let $R(x, y)$ have the reading x likes y . $R = \{\langle d, a \rangle, \langle a, b \rangle, \langle a, d \rangle\}$. Let $A = \{a, d\}$ be the animate objects in the domain $D = \{a, b, c, d\}$. We stipulate that $R \subseteq A \times D$, i.e. the first argument of R must be animate.
- b. Let $P \subseteq D \times D$ have the reading x belongs to y . In the given example, $P = \{\langle a, c \rangle, \langle d, b \rangle\}$.

Exercise 10, page 19:

- a. transitive
- b. transitive

- c. not transitive
- d. transitive (identity is an equivalence relation)
- e. probably not transitive in general
 Note: This case is controversial and it depends on what kind of similarity one has in mind. In cases of so-called Sorites paradoxes similarity does not seem to be transitive. Take for example, *having a similar color*. Color a might be similar to color b and b similar to c , but perhaps a is no longer considered similar to c . (Think of a smooth transition from red to orange.)
- f. not transitive (the mother of the mother of x is the grandmother of x)
- g. transitive (this is the identity relation, which is an equivalence relation)

Exercise 11, page 19:

- a. not reflexive, symmetric, not antisymmetric, not Euclidean, not transitive
- b. reflexive, symmetric, not antisymmetric, Euclidean, transitive
 Note: 'sameness' is understood in the sense of an equivalence relation, but there might be weaker readings that are not Euclidean and not transitive.
- c. not reflexive, not symmetric, antisymmetric, not Euclidean, transitive
- d. not reflexive, not symmetric, antisymmetric, not Euclidean, transitive
- e. reflexive, not symmetric, antisymmetric, not Euclidean, transitive
- f. not reflexive, not symmetric, not antisymmetric, not Euclidean, not transitive
- g. not reflexive, symmetric, not antisymmetric, not Euclidean, not transitive

Exercise 12, page 19: Any relation based on a strict and precise understanding of 'sameness' is an equivalence relation, e.g.: x has the same age as y , x has the same birthday as y , x and y have the same number of pets, x and y have the same number of children. Identity is also an equivalence relation.

Exercise 13, page 20:

- a. Yes, it even *must* contain cycles, because a preorder is reflexive. Hence, in the graph representation of the relation every node points to itself, which is a cycle.
- b. No, the depicted relation is not a preorder. By transitivity from $R(a,b)$ and $R(b,c)$ it would follow that $R(a,c)$, but there is no link from a to c in the picture (only in the opposite direction).

Exercise 14, page 26:

- a. total, not surjective, injective since \mathbb{N} denotes the set of positive integers (not injective if we include negative numbers)
- b. total, not surjective, injective
- c. total, not surjective, injective
Note: not total when 0 is included because $1/0$ is not defined (whether \mathbb{N} contains 0 or not varies from author to author; usually it doesn't)
- d. total, not surjective, not injective
- e. total, not surjective, not injective
- f. total, surjective, injective, bijective (identity function)
- g. total, not surjective, injective
- h. total, surjective, not injective
- i. not total, not surjective, not injective

Exercise 15, page 26:

- a. $f(x) = 2x$
- b. There is strictly speaking no inverse function, because the square root can be both positive and negative ($-2^2 = 4$ and $2^2 = 4$). By convention often the positive square root $f(x) = \sqrt{x}$ is taken as the inverse of the power function.
- c. $f(x) = x$
- d. $f(x) = x^2$
- e. not injective, so the inverse is a relation not a function
- f. $f = \{\langle \text{Thomas}, \text{Ana} \rangle, \langle \text{Peter}, \text{Teresa} \rangle, \langle \text{Maria}, \text{Klaus} \rangle\}$

Exercise 16, page 26:

- a. not a function; inverse: the function from the bearers of a Turkish proper names to their name (in the ideal case where every name bearer has only one name)
Note: At least unofficially people can have two names, and then there is no inverse function. The set of Turkish proper names might not be well-defined.
- b. not a function; the relation between a Portuguese sentence to its possible translations is not a function; the inverse relation is also not a function
- c. function; inverse: the function from all passport numbers to the respective owner

- d. not a function; the relation between a grammatically-well formed English sentence to its meaning is (usually) one to many
- e. not a function; one owner can have many dogs

Exercise 17, page 27: Let $R = \{\langle \text{Ana}, \text{Ana} \rangle, \langle \text{Pedro}, \text{Pedro} \rangle, \langle \text{Mustafa}, \text{Mustafa} \rangle, \langle \text{Joe}, \text{Joe} \rangle, \langle \text{Lisa}, \text{Lisa} \rangle, \langle \text{Ana}, \text{Pedro} \rangle, \langle \text{Ana}, \text{Mustafa} \rangle, \langle \text{Pedro}, \text{Ana} \rangle, \langle \text{Pedro}, \text{Mustafa} \rangle, \langle \text{Pedro}, \text{Joe} \rangle, \langle \text{Mustafa}, \text{Joe} \rangle, \langle \text{Mustafa}, \text{Lisa} \rangle, \langle \text{Lisa}, \text{Ana} \rangle, \langle \text{Lisa}, \text{Pedro} \rangle, \langle \text{Lisa}, \text{Mustafa} \rangle, \langle \text{Lisa}, \text{Joe} \rangle\}$. Then:

$$f(x, y) = \begin{cases} 1 & \text{if } \langle x, y \rangle \in R \\ 0 & \text{otherwise} \end{cases}$$

Exercise 18, page 27: 1.) $f(x) = 3x$, fixed point 0: $f(0) = 0$. 2.) $f(x) = x^x$, fixed point 1: $f(1) = 1^1 = 1$.

Exercise 19, page 27:

- a. $f(x) = \begin{cases} 1 & \text{if } x \text{ is a native speaker of German,} \\ 0 & \text{otherwise} \end{cases}$
- b. $1_A(x) = \begin{cases} 1 & \text{if } x = 1, x = 0, \text{ or } x = -1, \\ 0 & \text{otherwise} \end{cases}$
- c. $f(x) = \begin{cases} 1 & \text{if } x = \langle a, b \rangle \text{ such that } a \text{ says 'Hi!' to } b, \\ 0 & \text{otherwise} \end{cases}$
- d. $f(x) = \begin{cases} 1 & \text{if } x \text{ is a raining event,} \\ 0 & \text{otherwise} \end{cases}$
- e. $A := \{1, 2, 3, 4, 5\}$
 $1_A(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{otherwise} \end{cases}$

Note: The original formulation of A was deliberately obfuscated and B is not needed at all.

f. $1_{(A \cup B)}(x) = \begin{cases} 1 & \text{if } x \in A \text{ or } x \in B, \\ 0 & \text{otherwise} \end{cases}$

Or: $f(x) = \begin{cases} 1 & \text{if } x \in \{a, b, c, d, f\}, \\ 0 & \text{otherwise} \end{cases}$

g. $f(x) = \begin{cases} 1 & \text{if } x = \langle a, b, c, d, e \rangle \text{ such that} \\ & a \text{ buys } b \text{ from } c \text{ at price } d \text{ at time } e, \\ 0 & \text{otherwise} \end{cases}$

$$h. f(x) = \begin{cases} 1 & \text{if } x \text{ is a raven and } x \text{ is not black,} \\ 0 & \text{otherwise} \end{cases}$$

Exercise 20, page 27:

- | | |
|-------------------|-------------------|
| a. function | f. not a function |
| b. function | g. not a function |
| c. not a function | h. function |
| d. function | i. function |
| e. function | |

Exercise 21, page 28:

- a. no, it's a relation
b. no, it's a relation

Chapter 4

Exercise 39, page 104:

- a. $(\lambda x.x)a \xrightarrow{\beta} a$
- b. $((\lambda xyz.(Pz\ yx)c)a)b \xrightarrow{\beta} ((\lambda yz.(Pzyc)a)b) \xrightarrow{\beta} (\lambda z.(Pzac)b) \xrightarrow{\beta} (Pbac)$
- c. $((\lambda x\lambda y.(R\ xy) \wedge R(yx))a)b \xrightarrow{\beta} ((\lambda y.(R\ ay) \wedge (R\ ya))b) \xrightarrow{\beta} (R\ ab) \wedge (R\ ba)$
- d. $((\lambda P\lambda x.Px \vee \neg Px)(\lambda x.x = x))a \xrightarrow{\alpha} ((\lambda P\lambda y.Py \vee \neg Py)(\lambda x.x = x))a \xrightarrow{\beta} ((\lambda y.(\lambda x.x = x)y \vee (\neg(\lambda x.x = x))y))a \xrightarrow{\beta} ((\lambda y.y = y \vee \neg(y = y))a) \xrightarrow{\beta} a = a \vee \neg(a = a)$
- e. $((\lambda P^{eet}.Px)(\lambda x\lambda y.x = y))a \xrightarrow{\alpha} ((\lambda Pz.Pz)(\lambda x\lambda y.x = y))a \xrightarrow{\beta} ((\lambda z.(\lambda x\lambda y.x = y)z)a) \xrightarrow{\beta} ((\lambda z\lambda y.z = y)a) \xrightarrow{\beta} \lambda y.a = y$
- f. $\lambda x(Pa) \xrightarrow{\eta} Pa$
- g. $(\lambda P(Pxa))(\lambda x\lambda y.Qxy) \xrightarrow{\alpha} (\lambda P(Pza))(\lambda x\lambda y.Qxy) \xrightarrow{\beta} (((\lambda x\lambda y.Qxy)z)a) \xrightarrow{\beta} ((\lambda y.Qzy)a) \xrightarrow{\beta} Qza$
- h. $((\lambda PQ.\forall x[Px \rightarrow Qx])Q_0)P_0 \xrightarrow{\beta} (\lambda Q.\forall x[Q_0x \rightarrow Qx])P_0 \xrightarrow{\beta} \forall x[Q_0x \rightarrow P_0x]$



Index

- Łukasiewicz, 35
- abduction, 58
- abstraction, 103
- actualism, 84
- adder, 62
- adjunction, 32
- adverb, 90
- affirming the consequent, 56
- Ajdukiewicz, 105
- aleph null, 6
- all, 5, 7
- alpha conversion, 102, 104
- analysis
 - logical, 42
- anaphora, 87
- Andrews, 95, 117
- argument
 - deductive, 50, 55, 91
 - good, 57
 - scheme, 55
- arity, 13
- assignment, 70
- axiom of choice, 116
- axiom system, 45
- Bar-Hillel, 105
- Barwise, 118
- base
 - for PC, 40
- belief, 88
- Bentham, 88
- beta conversion, 102
- beta reduction, 103
- biconditional, 32, 38
- biimplication, 32
- binary numbers, 62
- bisubjunction, 32
- Bohr, 33
- Bourbaki, 3
- branch
 - closed, 48
 - open, 48
- calculus, 102
- Cantor, 6
- cardinality, 6, 94
- Carpenter, 119
- Cartesian Product, 14
- categorial grammar, 105, 119
- characteristic function, 25
- characterizability, 83
- Chomsky, 115
- Church, 22, 97, 98, 117
- Church-Rosser theorem, 116
- Combinatory Categorial Grammar,
 - 115, 119
- Compactness

- of PC, 61
- compactness, 97
 - of FOL, 94
 - of HOL, 117
- complement, 6
- completeness
 - of FOL, 94
 - of HOL, 97, 117
 - of PC, 61
- compound category
 - in CG, 105
- concatenation, 105
- conditional, 32, 38, 57
 - converse, 39
- confluency, 116
- conjunction, 32, 38, 103
 - asymmetric, 89
 - of NPs, 90
- connective, 32, 66
- consistency, 43
- constant
 - propositional, 31, 71
- contingency
 - logical, 42
- continuum, 6
- continuum hypothesis, 6, 116
- contradiction, 43, 57
- contraposition, 55
- Cooper, 118
- corollary, 46
- countable, 6, 8, 94
- counter-model, 48, 79
- counterfactual conditional, 90
- credibility, 58
- critical thinking, 63
- Curry, 118
- decidability
 - of PC, 61
- deduction
 - natural, 45
- deductive closure, 51
- definability, 83
- definite description, 82
 - definition, 34
 - recursive, 3
- DeMorgan, 50
- denumerable, 6, *see* countable
- denying the antecedent, 56
- description, 3, 82
- diamond property, 116
- difference, 6
- disjunction, 32
 - exclusive, 32, 38
 - inclusive, 32, 38
- domain
 - non-empty, 71
- enumeration, 2
- equivalence, 32
 - of wffs, 43
 - vs. biconditional, 43
- equivalence relation, 17
- eta conversion, 102, 104
- eta reduction, 103
- event, 72
- every, 83
- ex falso quod libet, 57
- exclusive OR, 32
- existence, 55, 84
- expletive it, 71
- expressive power, 83
- extension, 4, 14, 15
- extensionality principle, 4
- factivity, 89
- fallacy, 56
- Falsum function, 39
- Feys, 118
- first-order modal logic, 89
- Frege, 4, 28
- function, 20
 - bijjective, 24, 25
 - characteristic, 25
 - indicator, 25
 - injective, 23, 24
 - inverse, 24
 - partial, 20

surjective, 23
 total, 20
 functional application, 107
 future, 88
 God, 55, 84
 hammer, 42
 Heim, 28
 Henkin, 100, 117
 Hodges, 63, 95
 horseshoe, 32
 HPSG, 115
 identity, 44, 73
 between sets, 4
 identity of indiscernibles, 73
 if and only if, 32
 iff, 32
 implication, 32
 indexical, 87
 indicator function, 25
 indiscernibility of identicals, 73
 induction, 58
 infinite chain, 56
 infinity, 6
 intension, 4
 intensionality, 88
 interdefinability
 of quantifiers and identity, 101
 of truth functions, 39, 57
 intersection, 5
 introspection principle, 89
 inverse function, 24
 inverse relation, 15
 iota operator, 82, 86
 iota quantifier, 82, 86
 Isabelle, 102
 junctor, 32
 junctor main, 34
 KK principle, 89
 Kratzer, 28
 Kripke, 86
 lambda calculus, 22, 102
 Lambek calculus, 119
 Leibniz' Law, 73, 101
 lemma, 46
 Leo-II, 102
 Lewis, 105
 Lexical Functional Grammar, 115
 logic
 'informal', 63
 logical consequence, 58
 Löwenheim-Skolem theorem, 94
 main junctor, 34, 40
 many-sorted logic, 84
 material equivalence, 32
 material implication, 32, 57
 maximum, 56
 membership, 5
 metatheorem, 46
 minimalism, 115
 minimum, 56
 modal logic, 89
 model
 generalized Henkin, 100, 116
 intended, 44
 of FOL, 71
 of HOL, 100, 116
 standard, 100
 modification, 90, 115
 modus ponens, 55
 modus tollens, 55
 monotonicity, 57, 59
 Montague, 85, 89, 119
 Moortgat, 119
 Morrill, 119
 most, 8
 Mostowski, 28, 118
 murder, 119
 name, 86
 natural deduction, 45
 negation, 32, 38, 41
 double, 50
 negative introspection, 89

Newton, 42
 no, 7, 8
 non-classical logic, 57
 nonconditional, 39
 normalization theorem, 116
 notation
 polish, 35
 numerals, 8

 one-on-one, 23
 onto, 23
 ontological proof, 55, 84
 open formula, 69
 order
 partial, 17
 preorder, 16
 quasi-order, 16
 total, 17
 ordered pair, 13
 ordered tuple, 13

 paradox of the material implication,
 57
 Partee, 28
 partial order, 17
 partiality, 82
 past, 88
 Peirce stroke, 32, 38
 plausibility, 58, 59
 Polish notation, 35
 positive introspection, 89
 possibilism, 84
 possibility
 logical, 41
 powerset, 8
 predication, 73
 preference, 20
 preorder, 16, 20, 56, 59
 presupposition, 5
 projection, 39
 proof theory, 45
 proper name, 86
 proposition
 in mathematics, 46

 propositional attitude, 89
 prove, 48

 quantification
 first-order, 73
 higher-order, 83
 relativized, 83
 second-order, 73
 vacuous, 69
 quantifier, 5
 body, 111
 existential, 75
 first-order, 86
 generalized, 7, 28, 111–114, 118
 in FOL, 66
 relativized, 83
 restricted, 83
 restriction, 5, 111
 universal, 75, 76
 quantifier domain restriction, 83
 quantifier scope ambiguity, 114–115
 quasi-order, 16
 Quine dagger, 32

 recursion, 3
 reduction, 103
 reduction ad absurdum, 48
 reification, 115
 relation, 13
 antisymmetric, 16
 asymmetric, 16
 equivalence, 17
 Euclidean, 16
 irreflexive, 16
 part-of, 17
 reflexive, 16
 symmetric, 16
 total, 16
 transitive, 16
 rewrite system, 102
 Russell, 82

 satisfiability, 41, 43
 Schönfinkel, 118
 Schönfinkelization, 99, 101

- scope
 - of a quantifier, 69
- semantics
 - lexical, 111
 - of FOL, 70
 - of PC, 36
- semidecidability, 94
- sequent calculus, 45
- set, 1
 - abstraction, 2
 - empty, 3
- Sheffer stroke, 32, 38, 40
- situation, 72
- Smullyan, 94
- some, 7
- soundness
 - of a premise, 58
 - of an argument, 55
 - of FOL, 94
 - of PC, 61
- Steedman, 119
- subject
 - logical vs. grammatical, 71
- subjunction, 32
- subset, 5
- syllogism, 61
- syntactic category
 - in CG, 105
- syntax
 - of FOL, 65, 68
 - of HOL, 97
 - of PC, 31, 33
- tableaux, 45
 - for FOL, 74–79
 - for PC, 46–50
- Tarski, 63
- tautology, 41, 43
- tense, 72, 87
- ter Meulen, 28
- term, 66, 98
 - compound, 98
 - ground, 66
- theorem, 46
 - of FOL, 79
 - of PC, 52
- theorem prover, 102
- theory, 44
- three, 8
- time interval, 88
- todos, 7
- total order, 17
- total relation, 16
- TPS, 102
- transitive verb, 104
- tree
 - closed, 48
 - complete, 48
 - incomplete, 78
- Tree Adjoining Grammar, 115
- truth conditions, 85
- truth function, 39, 44, 57
- truth in a model, 72, 100
- truth preservation, 57
- truth table, 44
- truth-functionality, 89
- two-sorted logic, 84
- type, 97
- Type-Logical Grammar, 115, 119
- type-shifting, 112–114
- union, 5
- validity, 41, 43
- variable, 98
 - assignment, 70
 - binding, 69, 73
 - free vs. bound, 68
 - reuse, 69
- variant, 70, 73
- Venn diagram, 9
- verb, 85
 - ditransitive, 111, 114
 - intransitive, 111
 - transitive, 111, 113
- Verum function, 39
- VP-conjunction, 103
- Wall, 28

wff
of FOL, 66
of PC, 33