Undergraduate Topics in Computer Science

Mauricio Ayala-Rincón
Flávio L.C. de Moura

Applied Logic for Computer Scientists

Computational Deduction and Formal Proofs

Springer

Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

More information about this series at http://www.springer.com/series/7592

Mauricio Ayala-Rincón · Flávio L.C. de Moura

Applied Logic for Computer Scientists

Computational Deduction and Formal Proofs

Springer

Authors
Mauricio Ayala-Rincón
Departments of Mathematics and Computer Science
Universidade de Brasília
Brasília
Brazil

Flávio L.C. de Moura
Department of Computer Science
Universidade de Brasília
Brasília
Brazil

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-3-319-51651-6 ISBN 978-3-319-51653-0 (eBook)
DOI 10.1007/978-3-319-51653-0

Library of Congress Control Number: 2016963310

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Foreword

Despite the enormous progress in theorem proving technology in the last decade, formal verification is still a time consuming and human-intensive activity. This is particularly true in the case of formal verification of safety-critical and mission-critical algorithms. While elementary mathematical properties are usually elegantly formulated as simple statements about mathematical objects, correctness properties of algorithms are generally complex statements involving conditional statements, local definitions, and non-trivial data structures. The complexity involved in proving these correctness properties supports the myth that the application of formal methods in general, and formal verification in particular, requires highly trained mathematicians.

In reality, all proofs, proofs of elementary mathematical properties as well as correctness proofs of sophisticated algorithms, can be built on a relatively small set of basic deductive rules. These rules are the building blocks of modern interactive theorem provers and proof assistants. A common understanding of these rules and their deductive mechanisms is fundamental to a wider adoption of formal verification tools by software practitioners.

This book focuses on two styles of deductive rules: Natural deduction and Gentzen's calculus. The former is usually considered closer to the mathematical reasoning practice and yields a declarative style of proofs where conclusions follow from assumptions. The latter is closer to the goal-oriented proof construction mechanism used in many interactive theorem provers and yields a more procedural style of proofs. Despite their differences, both styles have the same deductive power. The book introduces classical propositional and predicate logics in both styles along with their completeness and correctness properties. It also briefly discusses constructive variants of these logics and their relations to simply typed lambda calculus. Finally, the last part of the book illustrates how these proof-theoretical concepts are used in the verification of algorithms specified in the Prototype Verification System (PVS).

The great Greek mathematician Euclid is said to have replied "there is no Royal Road to geometry" to King Ptolemy's request for an easier road to learning geometry.

Of course, the same answer applies to many other subjects in mathematics. In this book, the authors provide a gentle on-ramp to the study of logic and proof theory, with plenty of examples and exercises to help engineers and computer scientists on their journey towards the application of formal methods to real problems.

César A. Muñoz
Research Computer Scientist
NASA Langley Research Center
Hampton, VA
USA

Preface

This book compiles the course notes on logic we have been taught to computer science students at the *Universidade de Brasília* during almost ten years. We decided to provide students the essential fundaments on mathematical logic in an instrumental manner, restricting the discussion to only one relevant application of logic in computer science: logical deduction. Thus, the course notes provide the foundations of two different technologies to deal with logical deduction: natural deduction and Gentzen's sequent calculus. Natural deduction is studied for the propositional and predicate calculi highlighting elements from this deductive system that discriminate between constructive and classical deduction and culminating with a presentation of Gödel's completeness theorem. Gentzen's sequent calculus is presented as an alternative technology that is proved to be equivalent to natural deduction. As for natural deduction, in this alternative deductive technology we highlight the elements that discriminate between constructive and classical deduction.

The instrumental part of these notes consists of the operationalization of the deductive rules of Gentzen's sequent calculus in the context of proof assistants, using as computational framework the well-known *Prototype Verification System* (PVS). Connections between proof rules in this proof assistant and deductive rules in the sequent calculus are given and applications related with formal verification of properties of computational systems are illustrated through simple algebraic and algorithmic examples.

The principal motivation for the development of the notes is to offer undergraduate students of courses in engineering, computer science, and mathematics, the minimal theoretical background and most important, the minimal instrumental knowledge for the application of mathematical logic in the development of modern computer science. We found that this approach is adequate, since we detected that several students attending graduate courses on topics such as mathematical logic, type theory, proof theory and, in general on semantics of computation, despite being highly motivated, have a lack of the necessary basic knowledge and therefore are unable to apply elements of deductive logic that are used in nowadays computational artifacts. The essence of the problem is that they did not take logic

seriously since they did not realize that logic actually works as the cornerstone of several applications in computer science!

We are grateful to all the students, who have attended our courses and who have given us support as teaching assistants, and who have provided us valuable feedback, suggestions and corrections. In particular, we would like to thank Ariane Alves Almeida and Thiago Mendonça Ferreira Ramos for helping us in the PVS development of the sorting theory, which we have used to provide short course projects always related to the verification of simple algorithmic properties. This development is available in the web page that accompanies our notes: logic4CS.cic.unb.br. The authors are also grateful to Cesar Muñoz and other members of the Formal Methods group at NASA LaRC, as well as to Natarajan Shankar and Sam Owre from SRI International, the developers of PVS, for their kind support in issues related with the application and semantics of this proof assistant. This support was of great importance for the successful development of elaborated PVS theories by our research group at the *Universidade de Brasília* in which our students, now colleagues, André Luiz Galdino, Andréia Borges Avelar, Yuri Santos Rêgo and Ana Cristina Rocha-Oliveira played a paramount role. Despite all the received support, we would like to emphasize that all mistakes found in these course notes are our entire responsibility and, that we would be happy to receive all constructive feedbacks from the reader.

Last but not least, we would like to thank our families for understanding that academic work not only is done during working hours, but also requires hard work at home, and in particular to our wives, Mercedes and Tânia, to whom we dedicate this work.

Brasília D. F., Brazil — Mauricio Ayala-Rincón
September 2016 — Flávio L.C. de Moura

Contents

Introduction

Motivation

For decades, classical introductory textbooks presenting logic for undergraduate computer science students have been focused on the syntax and semantics of propositional and predicate calculi and related computational properties such as decidability and undecidability of logical questions. This kind of presentations, when given with the necessary formal details, are of great interest from the mathematical and computational points of view and conform a *sin equa non* basis for computer students interested in theory of computing as well as in the development of formal methods for dealing with robust software and hardware.

In addition to the unquestionable theoretical importance of these classical lines of presentation of the fundaments of logic for computer science, nowadays, it is of essential relevance to computer engineers and scientists for the precise understanding and mastering of the mathematical aspects involved in several deductive methods that are implemented and available in well-known modern proof assistants and deductive frameworks such as Isabelle, Coq, HOL, ACL2, PVS, among others. Only through the careful and precise use of this kind of computational tools, it is possible to assure the *mathematical correctness* of software and hardware that is necessary in order to guarantee the desired robustness of computer products.

Today, it is accepted that the software and hardware applied in critical systems, such as (sea, earth, air and space) human-guided or autonomous navigation systems, automotive hardware, medical systems and others, should have *mathematical certificates* of quality. But also it is clear for the computer science community that in all other areas in which computer science is being applied, this kind of formal verification is of fundamental importance (to try) to eliminate any possibility of harming or even offering a disservice to any user. These areas include financial, scheduling, administrative systems, games, among others; areas of application of computational systems in which users might be negatively affected by computational bugs. It is unnecessary to stress here that, nowadays, the society that is formed by the users of all these so-called *noncritical* systems is ready to complain

in a very organized manner against any detected bug; and, the computer engineers and scientists, who are involved in these developments, will be identified as those directly responsible. Indeed, nowadays, nobody accepts the old standard excuse given some years ago: "sorry, the issue was caused by an error of the computer."

The current presentation of logic for computer science and engineering focuses on the mathematical aspects subjacent to the deductive techniques applied to build proofs in both propositional and predicate logic. Derivations or proofs will be initially presented in the style of natural deduction and subsequently in the style of Gentzen's sequent calculus. Both these styles of deduction are implemented in several proof assistants and knowing how deductions are mathematically justified will be of great importance for the application of these tools in a very professional and effective manner. The correspondence between both styles of deduction will be discussed and simple computational applications in the PVS proof assistant will be given.

Examples

In order to explain the necessity of knowledge on formal deductive analysis and technologies for the adequate development of robust computer tools, we consider a classical piece of mathematics and computation that is implemented in the arithmetic libraries of the majority of modern computer languages. The problem is to compute the *greatest common divisor*, gcd for brevity, of pairs of integers that are not simultaneously zero.

In first place, the mathematical object that one wants to capture by an implementation is defined as below.

Definition 1 (*Greatest Common Divisor*). The greatest common divisor of two integer numbers i and j, that are not simultaneously equal to zero, is the greatest number k that divides both i and j.

Several observations are necessary before proceeding with the implementation of a simple program that effectively might correspond to the Definition 1. For instance, it is necessary to observe that the *domain* of the defined function gcd is $\mathbb{Z} \times \mathbb{Z} \backslash \{(0,0)\}$, while its *range* is $\mathbb{N}$. **Why?**

The first naive approach to implement gcd could be an *imperative* and *recursive algorithm* that checks, in a decreasing order, whether natural numbers divide both the integers i and j. The starting point of this algorithm is the natural number given by the smallest of the absolute values of the inputs: $min\{|i|, |j|\}$. But more sophisticated mathematical knowledge can be used in order to obtain more efficient solutions. For instance, one could apply a classical result that is attributed to Euclid and was developed more than two millennia ago.

Theorem 1 (Euclid 320-275 BC). $\forall m > 0, n \geq 0, \gcd(m,n)$ *equals to* $\gcd(m, n\,MOD\,m)$*, if* $n > 0$*,* m *otherwise; where* $n\,MOD\,m$ *denotes the remaining of the integer division of* n *by* m*.*

Observe that this result only will provide us a partial solution because, in this theorem, the domain is restricted to $(\mathbb{N}\backslash\{0\}) \times \mathbb{N}$, that is $\mathbb{N}^+ \times \mathbb{N}$, instead $\mathbb{Z} \times \mathbb{Z}\backslash\{(0,0)\}$, that is the domain of the mathematical object that we want to capture. Despite this drawback, we will proceed treating to capture a restricted version of the mathematical object gcd restricting its domain. In the end, $\gcd(i,j) = \gcd(|i|,|j|)$. **Why?**

A first easy observation is that Euclid's theorem does not provide any progress when $m > n$, because in this case $n\,\mathrm{MOD}\,m = n$. Thus, the theorem is of computational interest when $m \leq n$. The key point, in order to apply Euclid's theorem, is to observe that the remainder of the integer division between naturals n and m, $n\,\mathrm{MOD}\,m$ can be computed decreasing n by m, as many times as possible, whenever the result of the subtraction remains greater than or equal to m. This is done until a natural number smaller than m is reached. For instance, $27\,\mathrm{MOD}\,5$ equals to $((((27-5)-5)-5)-5)-5 = 2$. This procedure is possible in general, since for any integer k, $\gcd(m,n) = \gcd(m, n+km)$. **Why?**

Once the previously suggested procedure stops, one will have as first argument m, and as second argument a natural number, say $n - km$, that is less than m. The procedure can be repeated if one interchanges these arguments, since in general $\gcd(i,j) = \gcd(j,i)$. **Why?**

Following the previous observations, a first *attempt* to compute gcd restricted to the domain $\mathbb{N}^+ \times \mathbb{N}$ may be given by the *procedure* $\gcd_1$ presented in Algorithm 1.

The careful reader will notice that this first attempt fails because the restriction of the domain is not preserved by this specification; i.e., the first argument of this function may become equal to zero. For instance, for inputs 6 and 4 infinite recursive calls are generated by $\gcd_1$:

$$\begin{aligned}\gcd_1(4,6) &\rightarrow \gcd_1(4,2) \rightarrow \gcd_1(2,4) \rightarrow \gcd_1(2,2)\\ &\rightarrow \gcd_1(2,0) \rightarrow \gcd_1(\underline{0},2) \rightarrow \cdots\end{aligned}$$

procedure $\gcd_1(m : \mathbb{N}^+, n : \mathbb{N}) : \mathbb{N}$;
if $m > n$ **then**
 | $\overline{\gcd_1(n,m)}$
else
 | $\gcd_1(m, n-m)$
end

Algorithm 1: First attempt to specify gcd: procedure $\gcd_1$

In the end $\gcd_1$ fails because it is specified in such a manner that it never returns a natural number as answer, but instead recursive calls to $\gcd_1$.

Formally, the problem can be detected when trying to prove that the "function" specified as $\gcd_1$ is *well-defined*; i.e., to prove that the function is defined for all possible inputs of its domain. The attempt to prove well-definedness of $\gcd_1$ might be by nested induction on the first and second parameters of $\gcd_1$ as sketched below.

Induction Basis: Case $m = 1$. Notice that we start the induction from $m = 1$ since the type of m is $\mathbb{N}^+$. Trying to conclude by induction on n, two cases are to be considered: either $1 > n$ or $1 \leq n$. The case $1 > n$ gives rise to the recursive call $\gcd_1(0, 1)$ that has *ill-typed* arguments, since the first argument does not belong to the set $\mathbb{N}^+$ of positive naturals. The case $1 \leq n$ gives rise to the recursive call $\gcd_1(1, n-1)$, that is correctly typed since $n - 1 \geq 0$. But the attempt to conclude by induction on n fails.

Induction Step: Case $m > 1$.

Induction Basis: Case $n = 0$. $\gcd_1(m, 0) = \gcd_1(0, m)$ which is undefined, according to the analysis in the induction basis for m. To correct this problem, one needs to specify $\gcd(m, 0) = m$.
Induction Step: Case $n > 0$. This is done by analysis of cases:
Case $m > n$, $\gcd_1(m, n) = \gcd_1(n, m)$, that is well-defined by induction hypothesis, since $n < m$; that is, the first argument of $\gcd_1$ decreases.
Case $m \leq n$, $\gcd_1(m, n) = \gcd_1(m, n - m)$, that is well-defined by induction hypothesis, since the first argument remains the same and the second one decreases. Notice that in fact $n - m < n$, since in this step of the inductive proof m is assumed to be greater than zero. In addition, notice that $n - m$ has the correct type ($\mathbb{N}$), since $n - m \geq 0$.

Despite $\gcd_1$ is undefined for $m = 0$, one has that an eventual recursive call of the form $\gcd_1(0, n)$ produces a recursive call of the form $\gcd_1(0, n - 0)$, that as observed before might generate an infinite loop! Thus, a correct procedure should be specified in such a way that it takes care of this abuse on the restricted domain of the function gcd (restricted to the domain $\mathbb{N}^+ \times \mathbb{N}$) avoiding any possible recursive call with ill-typed arguments.

In the end this attempt to prove well-definedness of the procedure $\gcd_1$ fails, but it provides valuable pieces of information that are useful to correct the procedure. Several elements of logical deduction were applied in the analysis, among them:

- Application of the principle of mathematical induction;
- Contradictory argumentation (undefined arguments are defined by the specified function);
- Analysis of cases: $\gcd_1$ is well-defined for positive values of m and n because it is proved well-defined for a *complete* set of cases for m and n; namely, the case in which $m > n$ and the contrary case, that is the case in which $m > n$ does not hold, or equivalently, the case in which $m \leq n$.

Some of the mathematical aspects of this kind of logical analysis will be made precise through the next chapters of this book.

From the corrections made when attempting to prove well-definedness of $\gcd_1$ a new specification of the function gcd, called $\gcd_2$, is proposed in Algorithm 2.

A thoughtful revision of the attempt to proof well-definedness of the procedure $\gcd_1$ will provide a verification that this property is owned by the new specification $\gcd_2$. As before, the proof follows a nested induction on the first and second parameters of $\gcd_2$.

Induction Basis: Case $m = 1$. $\gcd_2(1, n)$ gives to cases according to whether $1 > n$ or $1 \leq n$, which are treated by nested induction on n.

> **Induction Basis**: Case $n = 0$. Since $1 > n$, the answer is 1.
> **Induction Step**: Case $n > 0$. This is the case in which $1 \leq n$, that gives rise to the recursive call $\gcd_2(1, n-1)$, that is well-defined by induction hypothesis for n.

Induction Step: Case $m > 1$.

> **Induction Basis**: Case $n = 0$. $\gcd_2(m, 0) = m$, which is correct.
> **Induction Step**: Case $n > 0$. This is done by analysis of cases:
> **Case** $m > n$, $\gcd_2(m, n) = \gcd_2(n, m)$, that is well-defined by induction hypothesis, since $n < m$; that is, the first argument of $\gcd_2$ decreases. Also, observe that since it is supposed that $n > 0$, this interchange of the arguments respects the type restriction on the parameters of $\gcd_2$.
> **Case** $m \leq n$, $\gcd_2(m, n) = \gcd_2(m, n-m)$, that is well-defined by induction hypothesis, since the first argument remains the same and the second one decreases. Notice that in fact $n - m < n$, since in this step of the inductive proof m is supposed to be greater than zero. Also, since $m \leq n$, $n - m \geq 0$, thus $n - m$ has the correct type $\mathbb{N}$.

The moral of this example is that the correction of the specification, from the point of view of well-definedness of the specified function, relies on the proof that it

```
procedure gcd2(m : N+, n : N) : N ;
if n = 0 then
|  m
else
|  if m > n then
|  |  gcd2(n, m)
|  else
|  |  gcd2(m, n − m)
|  end
end
```

Algorithm 2: Second attempt to specify gcd: procedure $\gcd_2$

is defined for all possible inputs. Therefore, in order to obtain correct implementations, it is essential to know how to develop proofs formally. Of course, it is much more complex to prove that in fact, $\gcd_2$ for inputs $(m,n) \in \mathbb{N}^+ \times \mathbb{N}$, correctly computes $\gcd(m,n)$.

Once well-definedness of $\gcd_2$ is guaranteed, becomes interesting proving that indeed this specification computes correctly the function gcd as given in the definition. For doing this one will require the application of Euclid's theorem as well as previously properties of gcd (that were highlighted with questions "Why?'s"):

1. For all integers i,j that are not simultaneously equal to zero, that is for all $(i,j) \in \mathbb{Z} \times \mathbb{Z}\backslash\{(0,0)\}$, $\gcd(i,j) \in \mathbb{N}$;
2. For all $(i,j) \in \mathbb{Z} \times \mathbb{Z}\backslash\{(0,0)\}$, $\gcd(i,j) = \gcd(|i|,|j|)$;
3. For all $(m,n) \in \mathbb{N}^+ \times \mathbb{N}$, and $k \in \mathbb{Z}$, $\gcd(m,n) = \gcd(m,n+km)$;
4. For all $(i,j) \in \mathbb{Z} \times \mathbb{Z}\backslash\{(0,0)\}$, $\gcd(i,j) = \gcd(j,i)$.

Exercise 1
Prove these four properties.

Notice that the third property is the one that justifies the second nested **else** case in the specification $\gcd_2$, since for the case in which $k = -1$, one has

$$\gcd(m,n) = \gcd(m,n-m)$$

Also, this justifies as well Euclid's theorem.

Exercise 2 (*)
Design an algorithm for computing the function gcd in its whole domain: $\mathbb{Z} \times \mathbb{Z}\backslash\{(0,0)\}$. Prove that your algorithm is well-defined and that is correct.

Hint: assuming the four properties and Euclid's theorem prove that $\gcd_2$ is algebraically correct in the following sense:

For all $(i,j) \in \mathbb{Z}^* \times \mathbb{Z}$, where $\mathbb{Z}^*$ denotes the non zero integers, $\gcd_2(|i|,|j|)$ computes a number $k \in \mathbb{N}$, such that

- k divides i,
- k divides j and
- For all $p \in \mathbb{Z}$ such that p divides both i and j, it holds that $p \leq k$.

The "**(*)**" in this exercise means that it is of a reasonably high level of complexity or that will require additional knowledge. But the readers do not need to worry if they cannot answer this exercise at this point, since the objective of the course notes is to bring the required theoretical background and minimum practice to be able to formally build mathematical certificates of computational objects, such as the inquired in this exercise.

Structure of the Book

The technology of computational formalization and verification is a mature area in both computer science and mathematical logic. This technology involves a great deal of precise mathematical knowledge about logical deduction and proof theory. These areas are originally placed in the setting of formalisms of mathematics and have been studied in detail since the earlier years of the last century by well-known mathematicians, from whom perhaps the most famous were David Hilbert, Luitzen Brouwer, Kurt Gödel, Alan Turing and Alonso Church, but since then, other researchers have provided well-known related results in computer science that are very useful and have been inspiring the development of proof assistants and formal methods tools (e.g., Gerhard Gentzen, Haskell Curry, Robert Floyd, Corrado Böhm, Robin Milner, Nicolaas de Bruijn, among others).

In this book we will focus on the mathematical technology of logical deduction for the most elementary logics that are the propositional logic and the logic of predicates. The focus will be on the calculi of deduction for these two logical systems according to two styles of deduction; namely, natural deduction and sequent calculus, and both deductive systems are contributions of the German mathematician Gerhard Gentzen. The motivation for restricting our attention to these two deduction styles and these two logical systems is that they are in the basis of all modern proof assistants.

In Chap. 1, we will present the *propositional logic* and its calculus in the style of *natural deduction*. We will present the syntax and semantics of this logical system and then we will prove that the deductive calculus in this natural style is *correct* and *complete*. *Correctness* is inherent to well-specified computer systems, in general, and in this setting it means that the deductive calculus deduces correct mathematical conclusions. *Completeness* means that all correct mathematical conclusions in the setting of this logical system can be deduced by the deductive calculus.

In Chap. 2, the propositional calculus is enriched with first-order variables which can be existentially and universally quantified by giving rise to the *logic of predicates* or first-order logic. This logical system provides a much more elaborated and expressive language, which corresponds to the basic logical language applied in most computational environments. To this end, the natural deductive calculus of the propositional logic will be enriched with rules for dealing with quantifiers and variables and, as for the propositional calculus, *correctness,* and *completeness* will be considered too.

In Chap. 3, the style of deduction known as Gentzen's *sequent calculus* will be considered and compared with the style of natural deduction, showing that both styles have the same deductive power. The presentation of this alternative calculus is of great computational interest because several proof assistants are based on this style of deduction.

In the first three chapters we will highlight several aspects about the constructive and classical logical systems making emphasis on the aspects related with the calculi associated with *minimal*, *intuitionistic*, and *classical* logic.

In Chap. 4, we will discuss in detail how proof commands of the well-known proof assistant prototype verification system PVS are related to deductive rules of the formal calculi as well as how inductive proofs and equational reasoning are conducted in the practice. In Chap. 5 applications of the deductive mechanisms will be studied through simple case studies formalized in PVS.

Chapter 1
Derivation and Proofs in the Propositional Logic

1.1 Motivation

The most elementary logical system of some practical interest is the propositional logic. In this system is it possible to express logical consequences, conjunction, disjunction, and negation of propositions. A proposition is a statement about something. For instance, one can state that "it is hot weather," that "it is raining" or that "driving is dangerous." Constructions with the language of propositional logic allow to express several combinations of statements. For instance using conjunction, disjunction, negation and implication, one obtains sentences such as:

"it is raining" and "driving is dangerous";
"it is raining" and "it is hot weather";
"it is raining" implies "driving is dangerous";
"it is hot weather" implies "driving isn't dangerous";
"it isn't raining" or "it isn't hot weather".

A great deal of knowledge can be expressed through the language of propositional logic and reasoning in this system is of great interest in several areas of computation such as programming and specification languages, relational data processing, design of digital circuits, etc.

1.2 Syntax of the Propositional Logic

The language of propositional logic, that is the language of propositional formulas, consists basically of propositional variables and logical connectives.

M. Ayala-Rincón and F.L.C. de Moura, *Applied Logic for Computer Scientists*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-51653-0_1

Table 1.1 Inference rules for the construction of well-formed propositional formulas: the calculus $\mathcal{C}_{pf}$

$$\frac{}{p}\ (\text{AXIOM VAR}),\ \text{if } p \in V \qquad \frac{}{\top}\ (\text{AXIOM } \top) \qquad \frac{}{\bot}\ (\text{AXIOM } \bot)$$

$$\frac{\phi}{(\neg\phi)}\ (\text{NEGATION}) \qquad \frac{\phi \quad \psi}{(\phi \wedge \psi)}\ (\text{CONJUNCTION})$$

$$\frac{\phi \quad \psi}{(\phi \vee \psi)}\ (\text{DISJUNCTION}) \qquad \frac{\phi \quad \psi}{(\phi \rightarrow \psi)}\ (\text{IMPLICATION})$$

Definition 2 (*Language of propositional logic*) The language of propositional logic is given by the set of words, denoted as Greek lower case letters, built from an enumerable set of propositional variables V, usually denoted by the later lower case letters of the Roman alphabet p, q, r, s, according to the following syntax:

$$\phi ::= V \ \ || \ \ \bot \ \ || \ \ \top \ \ || \ \ (\neg\phi) \ \ || \ \ (\phi \wedge \phi) \ \ || \ \ (\phi \vee \phi) \ \ || \ \ (\phi \rightarrow \phi)$$

A word built following these syntactical rules is called a *well-formed propositional formula*.

In the previous syntactical notation, ϕ represents an arbitrary well-formed propositional formula; $\phi ::= V$ means that variables in V are well-formed propositional formulas; $\phi ::= (\phi \rightarrow \phi)$ means that well-formed propositional formulas can be built from other two well-formed formulas of the language connecting them with the implication symbol $\rightarrow$ inside parenthesis. Finally, $||$ denotes choice.

This definition implies an inductive style of construction of well-formed propositional formulas: from well-formed formulas of the language others can be built using the connectives and parenthesis adequately according to the given syntax. This can be also formulated as a deductive calculus of construction of finite words over the alphabet of symbols in the set $V \cup \{\bot, \top, (,), \neg, \vee, \wedge, \rightarrow\}$; that will be denoted by $\mathcal{C}_{pf}$.

The application of the inference rules of the calculus $\mathcal{C}_{pf}$ in Table 1.1 in a top-down manner allows construction or derivation of well-defined formulas. In a bottom-up manner, application of the rules allows checking whether a word is in fact a well-defined formula. Both these manners of applying the deductive rules are illustrated in the following example.

Example 1 We can build the word $(((\neg p) \vee q) \rightarrow (\neg r))$ following the sequence of applications of the rules of this calculus below.

1. p is well-formed by (AXIOM VAR);
2. $(\neg p)$ is well-formed by (NEGATION) using 1;
3. q is well-formed by (AXIOM VAR);
4. $((\neg p) \vee q)$ is well-formed by (DISJUNCTION) using 2 and 3;
5. r is well-formed by (AXIOM VAR);
6. $(\neg r)$ is well-formed by (NEGATION) using 5;
7. $(((\neg p) \vee q) \rightarrow (\neg r))$ is well-formed by (IMPLICATION) using 4 and 6

Instead the previous sequential presentation of the application of the rules, this derivation can also be represented as a derivation tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{-}{p}\,\text{(AXIOM VAR)}}{(\neg p)}\,\text{(NEGATION)} \quad \dfrac{-}{q}\,\text{(AXIOM VAR)}}{((\neg p) \vee q)}\,\text{(DISJUNCTION)} \quad \dfrac{\dfrac{-}{r}\,\text{(AXIOM VAR)}}{(\neg r)}\,\text{(NEGATION)}}{(((\neg p) \vee q) \rightarrow (\neg r))}\,\text{(IMPLICATION)}$$

In a bottom-up manner, the calculus $\mathcal{C}_{pf}$ can be used to prove whether a finite word $\phi \in (V \cup \{\bot, \top, (,), \neg, \vee, \wedge, \rightarrow\})^*$ is a well-formed propositional formula. For instance, consider the word $((\neg p) \wedge (q \rightarrow (\neg p)))$. The unique applicable rule is (CONJUNCTION), whenever one is able to prove that both the formulas $(\neg p)$ and $(q \rightarrow (\neg p))$ are well-formed. The former is proved by bottom-up application of (NEGATION) followed by (AXIOM VAR), and the latter can be only be proved to be well-formed by application of (IMPLICATION), whenever q and $(\neg p)$ are provable to be well-formed formulas, that is possible by respective application of (AXIOM VAR) and reusing the proof for $(\neg p)$. This can also be presented as the following tree that should be read from bottom to top:

$$\dfrac{\dfrac{\dfrac{-}{p}\,\text{(AXIOM VAR)}}{(\neg p)}\,\text{(NEGATION)} \quad \dfrac{-}{q} \quad \dfrac{\dfrac{-}{p}\,\text{(AXIOM VAR)}}{(\neg p)}\,\text{(NEGATION)}}{((\neg p) \wedge (q \rightarrow (\neg p)))}\,\text{(IMPLICATION)}\,\text{(CONJUNCTION)}$$

Finally, consider the word $(q \rightarrow \neg p)))$ $\in (V \cup \{\bot, \top, (,), \neg, \vee, \wedge, \rightarrow\})^*$. One can prove that this word is not a well-formed propositional formula in the following way: firstly, the unique rule that applies is (IMPLICATION), whenever one is able to prove that q and $\neg p))$ are well-formed. The former is possible through application

of (AXIOM VAR), but no rule applies to prove that the latter word is a well-formed formula. In fact, except for (AXIOM VAR), that does not apply, all other rules expect as first symbol a left parenthesis. This failing attempt can also be represented as a tree.

$$\dfrac{\dfrac{}{q}\,(\text{AXIOM VAR}) \qquad \dfrac{}{\neg p))}\,(?)}{(q \rightarrow \neg p)))}\,(\text{IMPLICATION})$$

Definition 3 (*Derivation of well-formed propositional formulas through* $\mathcal{C}_{pf}$) Whenever a sequence of applications of rules in $\mathcal{C}_{pf}$ allows the construction of a formula ϕ, it is said that ϕ is deduced or derivable from $\mathcal{C}_{pf}$, that is denoted as $\vdash_{\mathcal{C}_{pf}} \phi$. Also, one says that ϕ is provable to be well-defined, or for brevity, only provable, by $\mathcal{C}_{pf}$. The set of well-formed propositional formulas, i.e., the set of all formula ϕ such that $\vdash_{\mathcal{C}_{pf}} \phi$, will be denoted by Prop.

The problem of verification of well-formedness of formulas is the question whether $\vdash_{\mathcal{C}_{pf}} \phi$ holds, given $\phi \in (V \cup \{\bot, \top, (,), \neg, \vee, \wedge, \rightarrow\})^*$.

Example 2 (*Continuing Example* 1) According to the derivations given in Example 1, one can say that $\vdash_{\mathcal{C}_{pf}} (((\neg p) \vee q) \rightarrow (\neg r))$ and $\vdash_{\mathcal{C}_{pf}} ((\neg p) \wedge (q \rightarrow (\neg p)))$, but not $\vdash_{\mathcal{C}_{pf}} (q \rightarrow \neg p)))$, also denoted as $\nvdash_{\mathcal{C}_{pf}} (q \rightarrow \neg p)))$.

Alternatively, the set Prop can be defined as follows:

Definition 4 The set Prop is the smallest set such that:

1. If p is a propositional variable then $p \in$ Prop, and $\bot, \top \in$ Prop;
2. If $\varphi, \psi \in$ Prop then $(\neg\varphi), (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi) \in$ Prop.

Question: Why such *smallest set* exists?

We finish this subsection with the definition of sub-formula:

Definition 5 (*Sub-formula*) The sub-formulas of the propositional formula ϕ are recursively defined as follows:

- ϕ is a sub-formula of itself;
- If $\phi = (\neg\psi)$ then all sub-formulas of ψ are sub-formulas of ϕ;
- If $\phi = (\psi \square \gamma)$, where $\square \in \{\wedge, \vee, \rightarrow\}$, then all sub-formulas of ψ and γ are sub-formulas of ϕ.
- In addition, if ψ is a sub-formula of φ and $\psi \neq \varphi$ then ψ is said to be a *proper* sub-formula.

1.3 Structural Induction

Before presenting the principle of structural induction, a few examples of natural induction are given.

Natural induction can be stated summarily as the following principle about properties of natural numbers:

Whenever one can prove that a specific property P of natural numbers

- holds for zero (Inductive Basis—**IB**), that is $P(0)$, and
- that from $P(n)$ one can prove that $P(n+1)$ (Inductive Step—**IS**),

one can deduce that P is a property of all naturals.

This principle has been applied in the introduction in order to prove well-definedness of the specification of the greatest common divisor, for instance.

Following this principle one can prove typical properties of natural numbers such as:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}, \text{ and}$$

$$\sum_{i=0}^{n} k^i = \frac{k^{n+1}-1}{k-1}.$$

The former is proved as follows:

IB $\sum_{i=1}^{0} i = \frac{0(0+1)}{2} = 0$, that is correct.
IS Supposing that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$, one has the following sequence of equalities, where $=_{IH}$ denotes the step in which the induction hypothesis is applied.

$$\sum_{i=1}^{n+1} i =$$

$$\sum_{i=1}^{n} i + (n+1) =_{IH}$$

$$\frac{n(n+1)}{2} + (n+1) =$$

$$(n+1)(\frac{n}{2}+1) =$$

$$(n+1)\frac{n+2}{2} = \frac{(n+1)(n+2)}{2}.$$

The latter is proved as follows:

IB $\sum_{i=0}^{0} k^i = \frac{k^{0+1}-1}{k-1} = 1$.
IS Supposing $\sum_{i=0}^{n} k^i = \frac{k^{n+1}-1}{k-1}$, one has the following sequence of equalities.

$$\sum_{i=0}^{n+1} k^i =$$

$$\sum_{i=0}^{n} k^i + k^{n+1} =_{IH}$$

$$\frac{k^{n+1}-1}{k-1} + k^{n+1} =$$

$$\frac{k^{n+1}-1+k^{n+1}(k-1)}{k-1} =$$

$$\frac{k^{n+2}-1}{k-1}.$$

In computation and mathematics elaborated inductive principles are needed. In particular, in computer science one needs to reason inductively about abstract structures such as symbols, terms, formulas, sets, sequences, graphs, trees, lists, among others, as well as about their computational representations as specific data structures or abstract data types. For instance, when proving a specific property about trees one uses the "size" of the structure that can be measured by the number of nodes, the height of the tree, or by other specific measurable structural characteristic of trees. Thus, the principle of structural induction can be stated as follows: let P be a property of a class of computational objects; whenever it is possible to prove that

- for each object a of this class, which has "the simplest" structure, $P(a)$ holds (**IB**) and,
- if supposing that for each object b with "simpler" structure than an object c the property holds, then $P(c)$ holds too,

one can conclude that P holds for all objects of this class.

To illustrate how this principle can be applied for the specific case of well-formed propositional formulas, we can compare formulas by the sub-formula relation. Thus, a proper sub-formula of a formula is considered structurally "simpler" than the formula. For instance, the formulas $((\neg p) \vee q)$, $(\neg p)$, $(\neg r)$, p and q are simpler than $(((\neg p) \vee q) \rightarrow (\neg r))$ and, variables and constants are the simplest well-formed propositional formulas.

Example 3 Now, we are able to prove properties of well-formed propositional formulas such as the fact that they have the same number of opening and closing parentheses (balanced parentheses).

IB The simplest formulas have no parentheses. Thus the number of opening and closing parentheses are equal.
IS Let ϕ be a well-formed propositional formula different from a variable, and from the constants. Then supposing that all sub-formulas of ϕ have balanced parentheses, different cases should be considered:

Case $\phi = (\neg\psi)$: since ψ is a sub-formula of ϕ, it has balanced parentheses. Thus the total number of opening and closing parentheses in ϕ are equal.

Case $\phi = (\psi \vee \gamma)$: since ψ and γ are sub-formulas of ϕ, they have balanced parentheses. Thus, since ϕ adds exactly one opening and one closing parenthesis, ϕ has balanced parentheses.

Case $\phi = (\psi \wedge \gamma)$: similar to the previous one.

Case $\phi = (\psi \rightarrow \gamma)$: similar to the previous one.

No other case is possible.

In this manner it is proved that well-formed propositional formulas have balanced parentheses.

Several demonstrations by structural induction will be presented in this book in order to prove that some properties hold for structures such as terms, formulas, and even derivations and proofs.

Formally, the (structural) induction principle for propositional formulas can be stated as follows:

Definition 6 (*Induction Principle for Prop*) Let P be a property over the set Prop. If

1. $P(\bot)$ and $P(\top)$ hold, and $P(p)$ holds, for any propositional variable p, and
2. $P((\neg\varphi))$ holds, whenever $P(\varphi)$ holds, and
3. $P((\varphi \wedge \psi))$, $P((\varphi \vee \psi))$, $P((\varphi \rightarrow \psi))$ hold, whenever $P(\varphi)$ and $P(\psi)$ hold,

then one can conclude that P holds for any propositional formula.

This principle can be adapted for several inductively defined structures, such as other classes of well-formed terms, lists, trees, formulas, proofs, etc., and will be done in this text in a straightforward manner without making explicit a new variation of the principle. In particular, we will use the principle in structures such as predicate formulas and proofs.

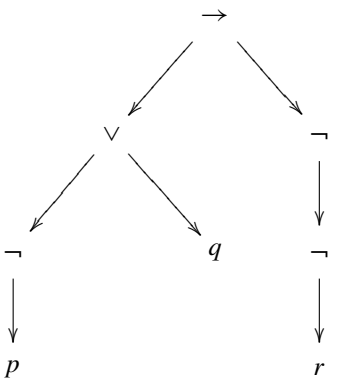
We can prove by structural induction that the structure of derivation trees corresponds to the structure of well-formed formulas. As derivations in $\mathcal{C}_{pf}$, well-formed formulas have a tree structure, as presented in the next definition.

Definition 7 (*Tree structure of formulas*) Inductively, a well-formed propositional formula ϕ is associated with a tree, denoted as T_ϕ, as follows:

- case ϕ equals a variable $p \in V$ or a constant in $\{\bot, \top\}$, the associated tree structure consists of a unique root node labeled with the variable p or the corresponding constant;
- case $\phi = (\neg\psi)$, for a well-formed formula ψ, the associated tree structure consists of a root node labeled with the connective $\neg$ and with unique sibling node the root of the tree associated with ψ;

- case $\phi = (\psi \Box \gamma)$, for $\Box \in \{\wedge, \vee, \rightarrow\}$ and ψ and γ well-formed propositional formulas, the associated structure consists of a root node labeled with the connective $\Box$ and with left sibling and right sibling nodes the root of the trees associated with ψ and γ, respectively.

Example 4 Consider the well-formed propositional formula $(((\neg p) \vee q) \rightarrow (\neg(\neg r)))$. Its associated tree structure is given as



Theorem 2 (Structure of proofs versus structure of propositional formulas) *The tree structure of any propositional formula is the same of its well-formedness proof.*

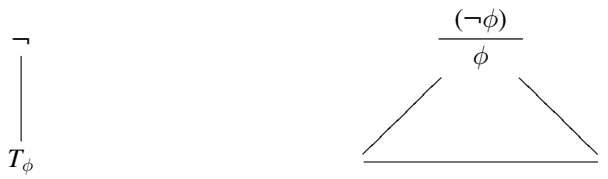
Proof The proof is by induction on the structure of propositional formulas.
IB The simplest formulas are propositional variables, and the constants $\bot$ and $\top$. Suppose $\diamond \in V \cup \{\bot, \top\}$, then the tree structure of $\diamond$, $T_\diamond$, consists of a unique root labeled with symbol $\diamond$, and the deduction tree of its well-formedness consists of a unique node for the application of (AXIOM) for variables, $\bot$ or $\top$, according to $\diamond$:

$\diamond$ $\qquad\qquad$ $\dfrac{}{\diamond}$ (AXIOM)

Tree $\qquad\qquad$ Proof

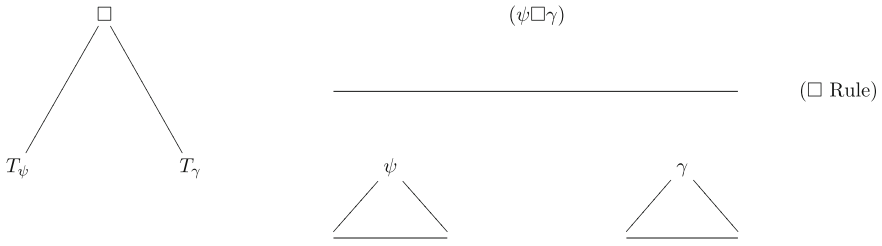
IS The proof is by case analysis.

Case the formula is a negation, $(\neg\phi)$, of a well-formed formula ϕ. By induction hypothesis, the tree structure of ϕ, T_ϕ, and its well-formedness proof coincide. The tree structure of $(\neg\phi)$ consists of a root labeled with the symbol $\neg$ and with sibling node the root of T_ϕ. The proof of well-formedness of $(\neg\phi)$ is the same as the proof of ϕ adding a derivation rule (NEGATION). Thus, the correspondence is completed associating the root of $T_{(\neg\phi)}$ with the root of the proof.



Tree Proof

Case $\phi = (\psi\Box\gamma)$, where $\Box \in \{\wedge, \vee, \rightarrow\}$ and both ψ and γ are well-formed propositional formulas. By induction hypothesis, the tree structure of the formulas ψ and γ, T_ψ and T_γ, respectively, coincide with the structure of their proofs of well-formedness. On the one side, the tree structure of ϕ consists of a root node labeled with symbol $\Box$ and with sibling nodes the roots of T_ψ and T_γ. On the other side the proof of well-formedness of ϕ consists of both the proofs for ψ and γ and an additional rule application, (CONJUNCTION) , (DISJUNCTION) or (IMPLICATION) , according to the symbol $\Box$.



Tree Proof

In this manner it is proved that the tree structure of well-formedness proofs and terms coincide. □

Exercise 3 Proof by structural induction that:

1. For any prefix s of a well-formed propositional formula ϕ, the number of open parentheses is greater than or equal to the number of closed parentheses in s.
2. Any proper prefix s of a well-formed propositional formula ϕ might not be a well-formed propositional formula. By "proper" we understand that s can not be equal to ϕ.

1.4 Natural Deductions and Proofs in the Propositional Logic

In this section, we will show that the goal of natural deduction is to deduce new information from facts that we already know, that we call *hypotheses* or *premises*. From now on, we will ignore external parentheses of formulas, whenever they do not introduce ambiguities. Suppose a set of formulas $S = \{\varphi_1, \varphi_2, \ldots, \varphi_n\}$ (for some $n > 0$) is given, and we want to know if the formula ψ can be obtained from S. We start with a simple reasoning, with $n = 2$: Suppose that the formulas φ_1 and φ_2 hold. In this case, we can conclude that the formula $\varphi_1 \wedge \varphi_2$ also holds (according to the usual meaning of the conjunction). This kind of reasoning is "natural" and can be represented by a nice mathematical notation as follows:

$$\frac{\phi_1 \qquad \phi_2}{\phi_1 \wedge \phi_2}$$

The formulas above the line are the *premises*, while the one below the line corresponds to the *conclusion*, i.e., the **new** information inferred from the premises.

Similarly, if we know that $\varphi_1 \wedge \varphi_2$ is true then so is φ_1, and also φ_2. This piece of reasoning can be represented by the following rules:

$$\frac{\phi_1 \wedge \phi_2}{\phi_1} \qquad\qquad \frac{\phi_1 \wedge \phi_2}{\phi_2}$$

With these three simple rules we can already *prove* a basic property of the conjunction: the commutativity, i.e., if $\varphi \wedge \psi$ then $\psi \wedge \varphi$. A proof is a tree whose leafs are premises and whose root is the conclusion. The internal nodes of the tree correspond to applications of the rules: any internal node is labeled by a formula that is the conclusion of the formulas labeling its ancestral nodes.

$$\dfrac{\dfrac{\phi \wedge \psi}{\psi} \qquad \dfrac{\phi \wedge \psi}{\phi}}{\psi \wedge \phi}$$

In the above tree, the hypothesis $\varphi \wedge \psi$ is used twice, and the conclusion is $\psi \wedge \varphi$. In other words, we have proved that $\varphi \wedge \psi \vdash \psi \wedge \varphi$. In general, we call an expression of the form $\varphi_1, \varphi_2, \ldots, \varphi_n \vdash \psi$ a *sequent*. The formulas before the symbol $\vdash$ are the premises, and the one after is the conclusion.

The system of natural deduction is composed by a set of inference rules. The idea is that each connective has an *introduction* and an *elimination* rule. Let us see how

it works for each connective. As we have seen, for the conjunction, the introduction rule is given by:

$$\frac{\phi_1 \qquad \phi_2}{\phi_1 \wedge \phi_2}\ (\wedge_i)$$

and two elimination rules:

$$\frac{\phi_1 \wedge \phi_2}{\phi_1}\ (\wedge_{e_1}) \qquad\qquad \frac{\phi_1 \wedge \phi_2}{\phi_2}\ (\wedge_{e_2})$$

The last two rules of elimination for the conjunction might be abbreviated as the unique rule:

$$\frac{\phi_1 \wedge \phi_2}{\phi_i \quad (i{=}1,2)}\ (\wedge_e)$$

The rules for implication are very intuitive: consider the following sentence

if it is raining then driving is dangerous

So, what one might conclude if it is raining? That driving is dangerous, of course. This kind of reasoning can be represented by an inference rule known as modus ponens (or elimination of the implication):

$$\frac{\phi \qquad \phi \to \psi}{\psi}\ (\to_e)$$

In order to introduce the implication $\phi \to \psi$ one needs to assume the premise of the implication, ϕ, and prove its conclusion, ψ. The (temporary) assumption ϕ is *discharged* once one introduces the implication, as depicted below:

$$\frac{\begin{array}{c}[\phi]^a \\ \vdots \\ \psi\end{array}}{\phi \to \psi}\ (\to_i)\ a$$

In this rule $[\phi]^a$ denotes the set of all leaves in the deduction of ψ where the formula ϕ was assumed. Thus, the label "a" is related with the set of all these assumptions in the derivation tree of ψ. And the application of the rule $(\to_i)$ uses this label "a"

to denote that all these assumptions are *closed* or *discharged* after the conclusion $\phi \to \psi$ is derived.

The $(\to_i)$ rule can also be applied without discharging any assumption: if one knows ψ then $\phi \to \psi$ holds for any ϕ. In this case application of the rule is labeled with $(\to_i)\ \emptyset$. The use of the *empty set* symbol as label is justified since a label "a", as explained before, is related with the set of all assumptions of ϕ in the derivation tree labeled with a. The intuition behind this reasoning can be explained by the following example: suppose that we known that "I cannot fall asleep," then both "I drink coffee implies that I cannot fall asleep" and "I don't drink coffee implies that I cannot fall asleep" hold. That is, using the previous notation, one obtains the following derivations, where r and p mean respectively, "I cannot fall asleep" and "I drink coffee":

$$\frac{r}{p \to r}\ (\to_i)\ \emptyset \qquad\qquad \frac{r}{\neg p \to r}\ (\to_i)\ \emptyset$$

Introduction of the implication without discharging premises can be also be derived from an application of the rule with discharge of assumption as below:

$$\dfrac{\dfrac{\dfrac{\psi \qquad [\phi]^a}{\psi \wedge \phi}\ (\wedge_i)}{\psi}\ (\wedge_e)}{\phi \to \psi}\ (\to_i)\ a$$

Application of rules with temporary assumptions can discharge either none or several occurrences of the assumed formula. For instance, consider the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\phi \to \phi \to \psi]^x \qquad [\phi]^y}{\phi \to \psi}\ (\to_e) \qquad [\phi]^z}{\psi}\ (\to_e)}{\phi \to \psi}\ (\to_i)\ z}{\phi \to \phi \to \psi}\ (\to_i)\ y}{(\phi \to \phi \to \psi) \to \phi \to \phi \to \psi}\ (\to_i)\ x$$

In the above example, the temporary assumption ϕ was partially discharged in the first application of the rule $(\to_i)$ since only the assumption of the formula ϕ with label z was discharged, but not the assumption with label y. A logical system that allows this kind of derivation is said to obey the *partial discharge convention*. The above

derivation can be solved with a complete discharge of the temporary assumption ϕ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\phi \to \phi \to \psi]^x \quad [\phi]^y}{\phi \to \psi}\,(\to_e) \quad [\phi]^y}{\psi}\,(\to_e)}{\phi \to \psi}\,(\to_i)\ \emptyset}{\phi \to \phi \to \psi}\,(\to_i)\ y}{(\phi \to \phi \to \psi) \to \phi \to \phi \to \psi}\,(\to_i)\ x$$

A logical system that forbids a partial discharge of temporary assumptions is said to obey the *complete discharge convention*. A comparison between the last two proofs suggests that partial discharges can be replaced by one complete discharge followed by vacuous ones. This is correct and so these discharge conventions play "little role in standard accounts of natural deduction," but it is relevant in *type theory* for the correspondence between proofs and λ-terms because "different discharge labels will correspond to different terms." For more details, see suggested readings and references on type theory (Chap. 6).

For the disjunction, the introduction rules are given by:

$$\dfrac{\phi_1}{\phi_1 \vee \phi_2}\,(\vee_{i_1}) \qquad\qquad \dfrac{\phi_2}{\phi_1 \vee \phi_2}\,(\vee_{i_2})$$

The first introduction rule means that, if ϕ_1 holds, or in other words, if one has a proof of ϕ_1, then $\phi_1 \vee \phi_2$ also holds, where ϕ_2 is any formula. The meaning of the rule $(\vee_{i_2})$ is similar. As for the elimination of conjunction rule $(\wedge_e)$, these two rules might be abbreviated as a unique one:

$$\dfrac{\phi_i \quad (i{=}1,2)}{\phi_1 \vee \phi_2}\,(\vee_i)$$

As another example of simultaneous discharging of occurrences of an assumption, observe the derivation for $\vdash (\phi \to ((\phi \vee \psi) \wedge (\phi \vee \varphi)))$ in which, by application of the rule of introduction of implication $(\to_i)$, two occurrences of the assumption of ϕ are discharged.

$$\dfrac{\dfrac{(\vee_i)\ \dfrac{[\phi]^u}{(\phi \vee \psi)} \qquad \dfrac{[\phi]^u}{(\phi \vee \varphi)}\ (\vee_i)}{((\phi \vee \psi) \wedge (\phi \vee \varphi))}\ (\wedge_i)}{(\phi \rightarrow ((\phi \vee \psi) \wedge (\phi \vee \varphi)))}\ (\rightarrow_i)\ u$$

The elimination rule for the disjunction is more subtle because from the fact that $\phi_1 \vee \phi_2$ holds, one does not know if ϕ_1, ϕ_2, or both ϕ_1 and ϕ_2 hold. Nevertheless, if a formula χ can be proved from ϕ_1 and also from ϕ_2, then it can be derived from $\phi_1 \vee \phi_2$. This is the idea of the elimination rule for the disjunction that is presented below. In this rule, the notation $[\phi_1]^a$ means that ϕ_1 is a temporary assumption, or a hypothesis. Note that the rule scheme $(\vee_e)$ is labeled with a, b which means that the temporary assumptions are discharged, i.e., the assumptions are closed after the rule is applied.

$$\dfrac{\phi_1 \vee \phi_2 \qquad \begin{matrix}[\phi_1]^a \\ \vdots \\ \chi\end{matrix} \qquad \begin{matrix}[\phi_2]^b \\ \vdots \\ \chi\end{matrix}}{\chi}\ (\vee_e)\ a, b$$

As an example consider the following reasoning: You know that both coffee and tea have caffeine, so if you drink one or the other you will not be able to fall asleep. This reasoning can be seen as an instance of the disjunction elimination as follows: Let p be a proposition whose meaning is "I drink coffee", q means "I drink tea" and r means "I cannot fall asleep." One can prove r as follows:

$$\dfrac{p \vee q \qquad (\rightarrow_e)\ \dfrac{[p]^a \qquad p \rightarrow r}{r} \qquad \dfrac{[q]^b \qquad q \rightarrow r}{r}\ (\rightarrow_e)}{r}\ (\vee_e)\ a, b$$

The above tree has 5 leafs:

1. the hypothesis $p \vee q$
2. the temporary assumption p
3. the fact $p \rightarrow r$ whose meaning is "if I drink coffee then I will not sleep."
4. the temporary assumption q
5. The temporary assumption $q \rightarrow r$ whose meaning is "if I drink tea then I would not fall asleep."

We need to assume p and q as "temporary" assumptions because we want to show that r is true independently of which one holds. We know that at least one of these propositions holds since we have that $p \vee q$ holds. Once the rule of elimination of disjunction is applied these temporary assumptions are discharged.

Exercise 4 Prove that $\phi \vee \psi \vdash \psi \vee \phi$, i.e., the disjunction is commutative.

For the negation, the rules are as follows:

$$\frac{\begin{array}{c}[\phi]^a \\ \vdots \\ \bot\end{array}}{\neg\phi}\ (\neg_i)\ a \qquad\qquad \frac{\phi \quad \neg\phi}{\bot}\ (\neg_e)$$

The introduction rule says that if one is able to prove $\bot$ (the absurd) from the assumption ϕ, then $\neg\phi$ holds. This rule discharges the assumption ϕ concluding $\neg\phi$. The elimination rule states that if one is able to prove both a formula ϕ and its negation $\neg\phi$ then one can conclude the absurd $\bot$.

Remark 1 Neither the symbol of negation $\neg$ nor the symbol $\top$ are necessary. $\top$ can be encoded as $\bot \to \bot$ and negation of a formula ϕ as $\phi \to \bot$. From this encoding, one can notice that rule $(\neg_e)$ is not essential; namely, it corresponds to an application of rule $(\to_e)$:

$$\frac{\phi \quad \phi \to \bot}{\bot}\ (\to_e)$$

Similarly, one can notice that rule $(\neg_i)$ is neither essential because it corresponds to an application of rule $(\to_i)$:

$$\frac{\begin{array}{c}[\phi]^a \\ \vdots \\ \bot\end{array}}{\phi \to \bot}\ (\to_i)\ a$$

The absurd has no introduction rule, but it has an elimination rule, which corresponds to the application of rule $(\neg_i)$ discharging an empty set of assumptions.

$$\frac{\bot}{\phi}\ (\bot_e)$$

The set of rules presented so far (summarized in Table 1.2) represents a fragment of the propositional calculus known as the *intuitionistic propositional calculus*, which

is considered as the logical basis of the constructive mathematics. The set of formulas derived from these rules are known as the *intuitionistic propositional logic*. Only the essential rules are presented, omitting for instance rules for introduction of disjunction to the right and elimination of conjunction to the right, since both the logical operators $\wedge$ and $\vee$ were proved to be commutative. Also derived rules are omitted. In particular, the rule $(\bot_e)$ is also known as the *intuitionistic absurdity rule*. Eliminating $(\bot_e)$ one obtains the *minimal propositional calculus*. The formulas derived from these rules are known as the *minimal propositional logic*.

Shortly, one can say that the constructive mathematics is the mathematics without the *law of the excluded middle* $(\varphi \vee \neg\varphi)$, denoted by (LEM) for short. In this theory one replaces the phrase "there exists" by "we can construct," which is particularly interesting for Computer Science. The law of the excluded middle is also known as the *law of the excluded third* which means that no third option is allowed (*tertium non datur*).

Remark 2 There exists a fragment of the intuitionistic propositional logic that is of great interest in Computer Science. This is known as the *implicational fragment of the propositional logic*, and it contains only the rules $(\rightarrow_i)$ and $(\rightarrow_e)$. The computational interest in this fragment is that it is directly related to type inference in the functional paradigm of programming. In this paradigm (untyped) programs can be seen as terms of the following language:

$$t ::= x \mid (t\ t) \mid (\lambda_x.t)$$

where x ranges over a set of term variables, $(t\ u)$ represents the *application* of the function t to the argument u, and $(\lambda_x.t)$ represents a function with parameter x and body t. The construction $(\lambda_x.t)$ is called an *abstraction*. Types are either atomic or functional and their syntax is given as:

$$\tau ::= \tau \mid \tau \rightarrow \tau$$

The type of a variable is annotated as $x : \tau$ and a context Γ is a finite set of type annotations for variables in which each variable has a unique type.

The simple typing rules for the above language are as follows:

$$\frac{\Gamma \vdash t : A \rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : B}\ (\textsc{App}) \qquad \frac{\Gamma \cup \{x : A\} \vdash t : B}{\Gamma \vdash (\lambda_x.t) : A \rightarrow B}\ (\textsc{Abs})$$

$$\frac{}{\Gamma \vdash x : A}\ (\textsc{Var}),\ x : A \in \Gamma$$

Notice that, if one erases the term information on the rule (App), one gets exactly the rule $(\rightarrow_e)$. Similarly, the type information of the rule (Abs) corresponds to the rule $(\rightarrow_i)$. The rule (Var) does not correspond to any rule in natural deduction, but to a single assumption $[A]^x$, that is a derivation of $A \vdash A$. As an example, suppose one wants to build a function that computes the sum of two natural numbers x and y. That

x and y are naturals is expressed through the type annotations $x : \mathbb{N}$ and $y : \mathbb{N}$. Thus, supposing one has proved that the function `add` has functional type $\mathbb{N} \to \mathbb{N} \to \mathbb{N}$ under context $\Gamma = \{x : \mathbb{N}, y : \mathbb{N}\}$, one can derive that $(\texttt{add}\ x\ y)$ has type $\mathbb{N}$ under the same context as follows:

$$(\text{App})\ \frac{\dfrac{\Gamma \vdash \texttt{add} : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \qquad \Gamma \vdash x : \mathbb{N}\ (\text{Var})}{\Gamma \vdash (\texttt{add}\ x) : \mathbb{N} \to \mathbb{N}} \qquad (\text{Var})\ \Gamma \vdash y : \mathbb{N}}{\Gamma \vdash ((\texttt{add}\ x)\ y) : \mathbb{N}}\ (\text{App})$$

The abstraction of the function projection of the first argument of a pair of naturals is built in this language as $(\lambda_x.(\lambda_y.x))$ and its type is derived as follows:

$$(\text{Abs})\ \frac{\dfrac{(\text{Var})\ \Gamma \vdash x : \mathbb{N}}{\{x : \mathbb{N}\} \vdash (\lambda_y.x) : \mathbb{N} \to \mathbb{N}}}{\vdash ((\lambda_x.(\lambda_y.x)) : \mathbb{N} \to \mathbb{N} \to \mathbb{N}}\ (\text{Abs})$$

For a detailed presentation on this subject, see the suggested readings and references on *type theory*.

The exclusion of (LEM) in the intuitionistic logic means that $(\varphi \vee \neg\varphi)$ holds only if one can prove either φ or $\neg\varphi$, while in *classical logic*, it is taken as an axiom. The classical logic can be seen as an extension of the intuitionistic logic, and hence there are sequents that are provable in the former, but not in the latter. The standard example of propositional formula that is provable in classical logic, but cannot be proved in intuitionistic logic is Peirce's law: $((\varphi \to \psi) \to \varphi) \to \varphi$.

It is relevant to stress here that in the classical propositional calculus the rule $(\bot_e)$ can discharge a non empty set of negative assumptions. This is not the case in the propositional intuitionistic calculus in which this rule can only be applied without discharging assumptions. Thus, the rules for the *propositional classical calculus* include a new rule for *proving by contradiction*, for short (PBC), in which after deriving the absurd one can discharge negative assumptions. Essentially, replacing $(\bot_e)$ by (PBC) one obtains the calculus of natural deduction for the classical propositional logic (see Table 1.3).

In general, in order to get classical logic, one can add to the set of rules of Table 1.2 one of the following rules, where the rules $(\neg\neg_e)$ and (LEM) are called respectively the rule of elimination of the double negation and rule for the law of middle excluded.

$$\frac{\neg\neg\phi}{\phi}\ (\neg\neg_e) \qquad \frac{}{\phi \vee \neg\phi}\ (\text{LEM}) \qquad \frac{\begin{array}{c}[\neg\phi]^a \\ \vdots \\ \bot\end{array}}{\phi}\ (\text{PBC})\ a$$

Table 1.2 Rules of natural deduction for intuitionistic propositional logic

Introduction rules	Elimination rules
$\dfrac{\varphi \quad \psi}{\varphi \wedge \psi}\ (\wedge_i)$	$\dfrac{\varphi \wedge \psi}{\varphi}\ (\wedge_e)$
$\dfrac{\varphi}{\varphi \vee \psi}\ (\vee_i)$	$\dfrac{\varphi \vee \psi \qquad \begin{array}{c}[\varphi]^u \\ \vdots \\ \chi\end{array} \qquad \begin{array}{c}[\psi]^v \\ \vdots \\ \chi\end{array}}{\chi}\ (\vee_e)\ u, v$
$\dfrac{\begin{array}{c}[\varphi]^u \\ \vdots \\ \psi\end{array}}{\varphi \to \psi}\ (\to_i)\ u$	$\dfrac{\varphi \quad \varphi \to \psi}{\psi}\ (\to_e)$
$\dfrac{\begin{array}{c}[\varphi]^u \\ \vdots \\ \bot\end{array}}{\neg\varphi}\ (\neg_i)\ u$	$\dfrac{\varphi \quad \neg\varphi}{\bot}\ (\neg_e)$
	$\dfrac{\bot}{\varphi}\ (\bot_e)$

In fact, any two of these rules can be proved from the third one. Assuming $(\neg\neg_e)$ one can prove (LEM) and (PBC):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\neg(\phi \vee \neg\phi)]^x \qquad \dfrac{[\phi]^u}{\phi \vee \neg\phi}\,(\vee_i)}{\bot}\,(\neg_e)}{\neg\phi}\,(\neg_i)\ u}{\phi \vee \neg\phi}\,(\vee_i) \qquad [\neg(\phi \vee \neg\phi)]^x}{\bot}\,(\neg_e)}{\neg\neg(\phi \vee \neg\phi)}\,(\neg_i)\ x}{\phi \vee \neg\phi}\,(\neg\neg_e)$$

$$\dfrac{\dfrac{\dfrac{[\neg\phi]^a}{\vdots}}{\dfrac{\bot}{\neg\neg\phi}\,(\neg_i)\ a}}{\phi}\,(\neg\neg_e)$$

One can also prove (LEM) and $(\neg\neg_e)$ from (PBC):

$$\dfrac{\dfrac{[\neg(\phi\vee\neg\phi)]^a \quad \dfrac{[\neg\phi]^b}{(\phi\vee\neg\phi)}\,(\vee_i)}{\bot}\,(\neg_e)}{\dfrac{\dfrac{\phi}{\phi\vee\neg\phi}\,(\vee_i)}{\bot}\,(\neg_e)}\,(\text{PBC})\ b \qquad [\neg(\phi\vee\neg\phi)]^a$$

$$\dfrac{\dfrac{\neg\neg\phi \quad [\neg\phi]^a}{\bot}\,(\neg_e)}{\phi}\,(\text{PBC})\ a$$

Finally, from (LEM) one can prove $(\neg\neg_e)$ and (PBC):

$$\dfrac{\text{(LEM)}\ \dfrac{}{\phi\vee\neg\phi} \quad \dfrac{\dfrac{[\neg\phi]^a \quad \neg\neg\phi}{\bot}\,(\neg_e)}{\phi}\,(\bot_e) \qquad [\phi]^b}{\phi}\,(\vee_e)\ a,b$$

$$\dfrac{\text{(LEM)}\ \dfrac{}{\phi\vee\neg\phi} \quad [\phi]^a \qquad \dfrac{\dfrac{[\neg\phi]^b}{\vdots}}{\dfrac{\bot}{\phi}}\,(\bot_e)}{\phi}\,(\vee_e)\ a,b$$

Table 1.3 includes the set of natural deduction rules for the classical propositional logic where our preference was to add the rule (PBC). Note that the rule $(\bot_e)$ can be removed from Table 1.3 because it can be deduced directly from (PBC) by an empty discharge.

Table 1.3 Rules of natural deduction for classical propositional logic

Introduction rules	Elimination rules
$\dfrac{\varphi \quad \psi}{\varphi \wedge \psi}\ (\wedge_i)$	$\dfrac{\varphi \wedge \psi}{\varphi}\ (\wedge_e)$
$\dfrac{\varphi}{\varphi \vee \psi}\ (\vee_i)$	$\dfrac{\varphi \vee \psi \qquad \begin{array}{c}[\varphi]^u \\ \vdots \\ \chi\end{array} \qquad \begin{array}{c}[\psi]^v \\ \vdots \\ \chi\end{array}}{\chi}\ (\vee_e)\ u, v$
$\dfrac{\begin{array}{c}[\varphi]^u \\ \vdots \\ \psi\end{array}}{\varphi \to \psi}\ (\to_i)\ u$	$\dfrac{\varphi \quad \varphi \to \psi}{\psi}\ (\to_e)$
$\dfrac{\begin{array}{c}[\varphi]^u \\ \vdots \\ \bot\end{array}}{\neg\varphi}\ (\neg_i)\ u$	$\dfrac{\varphi \quad \neg\varphi}{\bot}\ (\neg_e)$
	$\dfrac{\begin{array}{c}[\neg\varphi]^u \\ \vdots \\ \bot\end{array}}{\varphi}\ (\text{PBC})\ u$

Exercise 5 Prove that the rule $(\bot_e)$ is not essential, i.e., prove that this rule can be derived from the rules presented in Table 1.3.

There are several proofs that are useful in many situations. These proofs are pieced together to build more elaborated pieces of reasoning. For this reason, these proofs will be added as *derived rules* in our natural deduction system. The first one is for the introduction of the double negation: $\varphi \vdash \neg\neg\varphi$.

$$\dfrac{\dfrac{\varphi \qquad [\neg\varphi]^a}{\bot}\ (\neg_e)}{\neg\neg\varphi}\ (\neg_i)\ a$$

The corresponding derived rule is as follows:

$$\frac{\phi}{\neg\neg\phi}\ (\neg\neg_i)$$

Once, a derivation is done, new rules can be included to the set of applicable ones.

Another rule of practical interest is *modus tollens*, that states that whenever one knows that $\phi \to \psi$ and $\neg\psi$, $\neg\phi$ holds. For instance if we know both that "if Aristotle was Indian then he was Asian" and that "he wasn't Asian," then we have that "Aristotle wasn't Indian." *Modus tollens*, that is $(\neg\psi), (\phi \to \psi) \vdash (\neg\phi)$, can be derived as follows.

$$\dfrac{\dfrac{\dfrac{[\phi]^x \qquad (\phi \to \psi)}{\psi} (\to_e) \qquad (\neg\psi)}{\bot} (\neg_e)}{(\neg\phi)} (\neg_i)\ x$$

Thus, a new derived rule for *modus tollens* can be added:

$$\frac{(\neg\psi) \qquad (\phi \to \psi)}{(\neg\phi)}\ (\mathrm{MT})$$

Another useful derived rules are the *contrapositive* ones. In particular, proving an implication $(\phi \to \psi)$ by *contraposition* consists of proving $(\neg\psi \to \neg\phi)$ or vice versa. Thus, in order to use this reasoning mechanism, it is necessary to build derivations for $(\phi \to \psi) \vdash (\neg\psi \to \neg\phi)$ as well as for $(\neg\psi \to \neg\phi) \vdash (\phi \to \psi)$. A derivation for the former sequent is presented below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\phi \to \psi \qquad [\phi]^y}{\psi} (\to_e) \qquad [\neg\psi]^x}{\bot} (\neg_e)}{\neg\phi} (\neg_i)\ y}{\neg\psi \to \neg\phi} (\to_i)\ x$$

A derivation of the latter sequent is presented below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg\psi \to \neg\phi \qquad [\neg\psi]^y}{\neg\phi} (\to_e) \qquad [\phi]^x}{\bot} (\neg_e)}{\psi} (\mathrm{PBC})\ y}{\phi \to \psi} (\to_i)\ x$$

Table 1.4 Derived rules of natural deduction for propositional logic

$\dfrac{}{\varphi \vee \neg\varphi}$ (LEM)	
$\dfrac{\neg\neg\varphi}{\varphi}$ $(\neg\neg_e)$	$\dfrac{\varphi}{\neg\neg\varphi}$ $(\neg\neg_i)$
$\dfrac{\psi \to \varphi \quad \neg\varphi}{\neg\psi}$ (MT)	$\dfrac{\bot}{\varphi}$ $(\bot_e)$
$\dfrac{\varphi \to \psi}{\neg\psi \to \neg\varphi}$ (CP_1)	$\dfrac{\neg\varphi \to \neg\psi}{\psi \to \varphi}$ (CP_2)

Thus, new derived rules for contraposition, for short (CP), can be given as:

$$\frac{\phi \to \psi}{\neg\psi \to \neg\phi}\ (\text{CP}_1) \qquad \frac{\neg\psi \to \neg\phi}{\phi \to \psi}\ (\text{CP}_2)$$

A few interesting rules that can be derived from the natural deduction calculus (as given in Table 1.3) are presented in Table 1.4.

Definition 8 (*Formulas provable equivalent*) Let ϕ and ψ be well-formed propositional formulas. Whenever, one has that $\phi \vdash \psi$ and also that $\psi \vdash \phi$, it is said that ϕ and ψ are *provable equivalent*. This is denoted as $\phi \dashv\vdash \psi$.

Notice that $\phi \to \psi \dashv\vdash \neg\psi \to \neg\phi$.

Exercise 6 Build derivations for both versions of contraposition below.

a. $\neg\psi \to \phi \dashv\vdash \neg\phi \to \psi$ and
b. $\psi \to \neg\phi \dashv\vdash \phi \to \neg\psi$.

In the sequel, several examples are presented.

Example 5 (*Associativity of conjunction and disjunction*) Derivations of the associativity of conjunction and disjunction are presented.

- First, the associativity of conjunction is proved; that is, $(\phi \wedge (\psi \wedge \varphi)) \vdash ((\phi \wedge \psi) \wedge \varphi)$:

$$\dfrac{\dfrac{(\phi \wedge (\psi \wedge \varphi))}{\phi}\,(\wedge_e) \qquad \dfrac{\dfrac{(\phi \wedge (\psi \wedge \varphi))}{(\psi \wedge \varphi)}\,(\wedge_e)}{\psi}\,(\wedge_e)}{(\phi \wedge \psi)}\,(\wedge_i) \qquad \dfrac{\dfrac{(\phi \wedge (\psi \wedge \varphi))}{(\psi \wedge \varphi)}\,(\wedge_e)}{\varphi}\,(\wedge_e)}{((\phi \wedge \psi) \wedge \varphi)}\,(\wedge_i)$$

Exercise 7 As an exercise, prove that $((\phi \wedge \psi) \wedge \varphi) \vdash (\phi \wedge (\psi \wedge \varphi))$.

- Second, the associativity of disjunction is proved; that is, $(\phi \vee (\psi \vee \varphi)) \vdash ((\phi \vee \psi) \vee \varphi)$:

$$\dfrac{(\phi \vee (\psi \vee \varphi)) \qquad \dfrac{(\vee_i)\ \dfrac{[\phi]^x}{(\phi \vee \psi)}}{((\phi \vee \psi) \vee \varphi)} \qquad \begin{matrix}\nabla \\ ((\phi \vee \psi) \vee \varphi)\end{matrix}}{((\phi \vee \psi) \vee \varphi)}\,(\vee_e)\ x, y$$

where ∇ is the derivation below:

$$\dfrac{[(\psi \vee \varphi)]^y \qquad \dfrac{(\vee_i)\ \dfrac{[\psi]^u}{(\phi \vee \psi)}}{((\phi \vee \psi) \vee \varphi)} \qquad \dfrac{[\varphi]^u}{((\phi \vee \psi) \vee \varphi)}\,(\vee_i)}{((\phi \vee \psi) \vee \varphi)}\,(\vee_e)\ u, v$$

Exercise 8 As an exercise, prove that $((\phi \vee \psi) \vee \varphi) \vdash (\phi \vee (\psi \vee \varphi))$.

Exercise 9 Classify the derived rules of Table 1.4 discriminating those that belong to the intuitionistic fragment of propositional logic, and those that are classical. For instance, (CP_1) was proved above using only intuitionistic rules, which means that it belongs to the intuitionistic fragment.
Hint: to prove that a derived rule is not intuitionistic, one can show that using only intuitionistic rules, a strictly classical rule such as (PBC) , (LEM) or $(\neg\neg_e)$ can be derived.

Exercise 10 Check whether each variant of contraposition below is either an intuitionistic or a classical rule.

$$\dfrac{\neg\varphi \to \psi}{\neg\psi \to \varphi}\,(\mathrm{CP}_3) \qquad\qquad \dfrac{\varphi \to \neg\psi}{\psi \to \neg\varphi}\,(\mathrm{CP}_4)$$

Exercise 11 Similarly, check whether each variant of (MT) below is either an intuitionistic or a classical rule.

$$\dfrac{\varphi \to \neg\psi \qquad \psi}{\neg\varphi}\,(\mathrm{MT}_2) \qquad \dfrac{\neg\varphi \to \psi \qquad \neg\psi}{\varphi}\,(\mathrm{MT}_3) \qquad \dfrac{\neg\varphi \to \psi \qquad \neg\psi}{\varphi}\,(\mathrm{MT}_4)$$

Exercise 12 Using only the rules for the minimal propositional calculus, i.e. the rules in Table 1.2 without $(\bot_e)$, give derivations for the following sequents.

a. $\neg\neg\neg\phi \dashv\vdash \neg\phi$.
b. $\neg\neg(\phi \to \psi) \vdash (\neg\neg\phi) \to (\neg\neg\psi)$.
c. $\neg\neg(\phi \wedge \psi) \dashv\vdash (\neg\neg\phi) \wedge (\neg\neg\psi)$.
d. $\neg(\phi \vee \psi) \dashv\vdash (\neg\phi \wedge \neg\psi)$.
e. $\phi \vee \psi \vdash \neg(\neg\phi \wedge \neg\psi)$.
f. $\vdash \neg\neg(\phi \vee \neg\phi)$.

Exercise 13 Using the rules for the intuitionistic propositional calculus, that is the rules in Table 1.2, give derivations for the following sequents.

a. $(\neg\neg\phi) \to (\neg\neg\psi) \vdash \neg\neg(\phi \to \psi)$. Compare with item b of Exercise 12.
b. $\vdash \neg\neg(\neg\neg\phi \to \phi)$.

Exercise 14 **(*)** A propositional formula ϕ belongs to the *negative fragment* if it does not contain disjunctions and all propositional variables occurring in ϕ are preceded by negation. Formulas in this fragment have the following syntax.

$$\phi ::= (\neg v) \;\;||\;\; \bot \;\;||\;\; (\neg\phi) \;\;||\;\; (\phi \wedge \phi) \;\;||\;\; (\phi \to \phi), \quad \text{for } v \in V$$

Prove by induction on ϕ, that for any formula in the negative fragment there are derivations in the minimal propositional calculus for

$$\vdash \phi \leftrightarrow \neg\neg\phi$$

i.e., prove $\vdash \phi \to \neg\neg\phi$ and $\vdash \neg\neg\phi \to \phi$.

Exercise 15 Give deductions for the following sequents:

a. $\neg(\neg\phi \wedge \neg\psi) \vdash \phi \vee \psi$.
b. Peirce's law: $\vdash ((\phi \to \psi) \to \phi) \to \phi$.

Exercise 16 **(*)** Let Γ be a set, and φ be a formula of propositional logic. Prove that if φ has a classical proof from the assumptions in Γ then $\neg\neg\varphi$ has an intuitionistic proof from the same assumptions. This fact is known as Glivenko's theorem (1929).

Exercise 17 **(*)** Consider the *negative Gödel translation* from classical propositional logic to intuitionistic propositional logic given by:

- $\bot^n = \bot$
- $p^n = \neg\neg p$, if p is a propositional variable.
- $(\varphi \wedge \psi)^n = \varphi^n \wedge \psi^n$
- $(\varphi \vee \psi)^n = \neg\neg(\varphi^n \vee \psi^n)$
- $(\varphi \to \psi)^n = \varphi^n \to \psi^n$

Prove that if $\Gamma \vdash \varphi$ in classical propositional logic then $\Gamma^n \vdash \varphi^n$ in intuitionistic propositional logic.

Exercise 18 Prove the following sequent, the double negation of Peirce's law, in the intuitionistic propositional logic: $\vdash \neg\neg(((\phi \to \psi) \to \phi) \to \phi)$

1.5 Semantics of the Propositional Logic

Deduction and derivation correspond to mechanical inference of *truth*. All syntactic deductive mechanisms that we have seen in the previous section can be blindly followed in order to prove that a formula of the propositional logic "holds", but in fact there was not presented a semantical counterpart of the notion of *being provable*. In this section we present the simple semantics of propositional logic.

In propositional logic the two only possible *truth-values* are *True* and *False*, denoted by brevity as T and F. No other truth-values are admissible, as it is the case in several other logical systems (e.g., truth-values as *may be true*, *probably*, *don't know*, *almost true*, *not yet, but in the future*, etc.).

Definition 9 (*Truth-values of atomic formula and assignments*) In propositional logic the *truth-values* of the basic syntactic formula, that are $\bot$, $\top$ and variables in V, are given in the following manner:

- the truth-value of $\bot$ is F;
- the truth-value of $\top$ is T;
- the truth-value of a variable v in the set of variables V, is given through a propositional *assignment* function from V to $\{T, F\}$. Thus, given an assignment function $d : V \to \{T, F\}$, the truth-value of $v \in V$ is given by $d(v)$.

The truth-value assignment to propositional variables deserve special attention. First, an assignment is necessary because variables neither can be interpreted as true or false without having fixed an assignment. Second, only after one has an assignment, it is possible to decide whether (the truth-value of) a variable is either true or false. Finally, the truth-value of propositional variables exclusively depends of a unique given assignment function.

Once an assignment function is given, one can determine the truth-value or semantical interpretation of nonatomic propositional formulas according to the following inductive definition.

Definition 10 (*Interpretation of propositional formula*) Given an assignment d over the set of variables V, the truth-value or interpretation of a propositional formula φ is determined inductively as below:

i. If $\varphi = \bot$ or $\varphi = \top$, one says that φ is F or T, respectively;
ii. if $\varphi = v \in V$, one says that φ is $d(v)$;
iii. if $\varphi = (\neg\psi)$, then its interpretation is given from the interpretation of ψ by the truth-table below:

ψ	$\varphi = (\neg\psi)$
T	F
F	T

iv. if $\varphi = (\psi \vee \phi)$, then its interpretation is given from the interpretations of ψ and ϕ according to the truth-table below:

ψ	ϕ	$\varphi = (\psi \vee \phi)$
T	T	T
T	F	T
F	T	T
F	F	F

v. if $\varphi = (\psi \wedge \phi)$, then its interpretation is given from the interpretations of ψ and ϕ according to the truth-table below:

ψ	ϕ	$\varphi = (\psi \wedge \phi)$
T	T	T
T	F	F
F	T	F
F	F	F

vi. if $\varphi = (\psi \rightarrow \phi)$, then its interpretation is given from the interpretations of ψ and ϕ according to the truth-table below:

ψ	ϕ	$\varphi = (\psi \rightarrow \phi)$
T	T	T
T	F	F
F	T	T
F	F	T

According to this definition, it is possible to determine *the* truth-value of any propositional formula under a specific assignment. For instance, to determine that the formula $(v \rightarrow (\neg v))$ is false for a given assignment d for which $d(v) = T$, one can build the following *truth-table* according to the assignment of v under d and the inductive steps for the connectives $\neg$ and $\rightarrow$ of the definition:

v	$(\neg v)$	$(v \rightarrow (\neg v))$
T	F	F

Similarly, if d' is an assignment for which, $d'(v) = F$, one obtains the following *truth-table*:

v	$(\neg v)$	$(v \rightarrow (\neg v))$
F	T	T

Notice, that *the* interpretation of a formula depends on the given assignment. Also, although we are talking about *the* interpretation of a formula under a given

assignment it was not proved that, given an assignment, formulas have a unique interpretation. That is done in the following lemma.

Lemma 1 (Uniqueness of interpretations) *The interpretation of a propositional formula φ under a given assignment d is unique and it is either true or false.*

Proof The proof is by induction on the structure of propositional formulas.
IB In the three possible cases the truth-value is unique: for $\bot$ false, for $\top$ true and for $v \in V$, $d(v)$ that is unique since d is functional.
IS This is done by cases.

Case $\varphi = (\neg\psi)$. By the hypothesis of induction ψ is either true or false and consequently, following the item *iii.* of the definition of interpretation of propositional formulas, the interpretation of φ is univocally given by either false or true, respectively.

Case $\varphi = (\psi \vee \phi)$. By the hypothesis of induction the truth-values of ψ and ϕ are unique and consequently, according to the item *iv.* of the definition of interpretation of propositional formulas, the truth-value of φ is unique.

Case $\varphi = (\psi \wedge \phi)$. By the hypothesis of induction the truth-values of ψ and ϕ are unique and consequently, according to the item *v.* of the definition of interpretation of propositional formulas, the truth-value of φ is unique.

Case $\varphi = (\psi \rightarrow \phi)$. By the hypothesis of induction the truth-values of ψ and ϕ are unique and consequently, according to the item *vi.* of the definition of interpretation of propositional formulas, the truth-value of φ is unique. □

It should be noticed that a formula may be interpreted both as true and false for different assignments. Uniqueness of the interpretation of a formula holds only once an assignment is fixed. Notice, for instance that the formula $(v \rightarrow (\neg v))$ can be true or false, according to the selected assignment. If it maps v to T, the formula is false and in the case that it maps v to F, the formula is true.

Whenever a formula can be interpreted as true for some assignment, it is said that the formula is satisfiable. In the other case it is said that the formula is unsatisfiable or invalid.

Definition 11 (*Satisfiability and unsatisfiability*) Let φ be a propositional formula. If there exists an assignment d, such that φ is true under d, then it is said to be *satisfiable*. If there does not exist such an assignment, it is said that φ is *unsatisfiable*.

The semantical counterpart of derivability is the notion of being a *logical consequence*.

Definition 12 (*Logical consequence and validity*) Let $\Gamma = \{\phi_1, \ldots, \phi_n\}$ be a finite set of propositional formulas that can be empty, and φ be a propositional formula. Whenever for all assignments under which all formulas of Γ are true, also φ is true, one says that φ is a *logical consequence* of Γ, which is denoted as

$$\Gamma \models \varphi$$

When Γ is the empty set one says that φ is *valid*, which is denoted as

$$\models \varphi$$

Notice that the notion of validity of a propositional formula φ, corresponds to the nonexistence of assignments for which φ is false. Then by simple observations of the definitions, we have the following lemma.

Lemma 2 (Satisfiability versus validity)

i. Any valid formula is satisfiable.
ii. The negation of a valid formula is unsatisfiable

Proof i. Let φ be a propositional formula such that $\models \varphi$. Then given any assignment d, φ is true under d. Thus, φ is satisfiable.
ii. Let φ be a formula such that $\models \varphi$. Then for all assignments φ is true, which implies that for all assignments $(\neg\varphi)$ is false. Then there is no possible assignment for which $(\neg\varphi)$ is true. Thus, $(\neg\varphi)$ is unsatisfiable.

□

1.6 Soundness and Completeness of the Propositional Logic

The notions of *soundness* (or *correctness*) and *completeness* are not restricted to deductive systems being also applied in several areas of computer science. For instance, we can say that a sorting algorithm is sound or correct, whenever for any possible input, that is a list of keys, this algorithm computes as result a *sorted* list, according to some ordering which allows comparison of these keys. Unsoundness or incorrectness of the algorithm could happen, when for a specific input the algorithm cannot give as output a sorted version of the input; for instance, the algorithm can compute as output a unordered list containing all keys in the input, or it can omit some keys that appear in the input list, or it can include some keys that do not appear in the input list, etc. In the context of logical deduction, correctness means intuitively that all derived formulas are in fact semantically correct. Following our example, the sorting algorithm will be said to be complete, whenever it is capable to sort all possible input lists. An incomplete sorting algorithm may be unable to sort simple cases such as the cases of the empty or unitary lists, or may be unable to sort lists with repetitions. From the point of view of logical deduction, completeness can be intuitively interpreted as the capability of a deductive method of building proofs for all possible logical consequences.

1.6.1 Soundness of the Propositional Logic

The propositional calculus, as given by the rules of natural deduction presented in Table 1.3, allows derivation of semantically *sound* (or *correct*) conclusions. For instance, rule $(\wedge_i)$, allows a derivation for the sequent $\varphi, \psi \vdash \varphi \wedge \psi$, which is semantically correct because whenever φ and ψ are true, $\varphi \wedge \psi$ is true; that is denoted as $\varphi, \psi \models \varphi \wedge \psi$. The *correctness* of the propositional logic is formalized in the following theorem.

Theorem 3 (Soundness of the propositional logic) *If* $\Gamma \vdash \varphi$*, for a finite set of propositional formulas* $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$*, then* $\Gamma \models \varphi$*. This can be summarized as*

$$\Gamma \vdash \varphi \textit{ implies } \Gamma \models \varphi$$

And for the case of Γ *equal to the empty set, we have that provable theorems are valid formulas:*

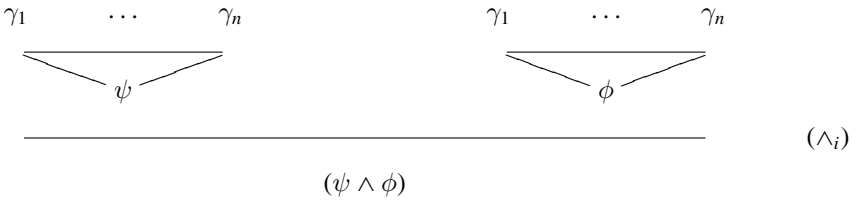
$$\vdash \varphi \textit{ implies } \models \varphi$$

Proof The proof is by induction on the structure of derivations. We will consider the last step of a derivation having as consequence the formula φ and as assumptions only formulas of Γ.

IB The most simple derivations are those that correspond to a simple selection of the set of assumptions that are derivations for sequents in which the conclusion is an assumption belonging to the set Γ; that is, $\gamma_1, \ldots, \gamma_i(=\varphi), \ldots, \gamma_n \vdash \varphi$. Notice that these derivations are correct since $\gamma_1, \ldots, \gamma_i(=\varphi), \ldots, \gamma_n \models \varphi$.

IS For the inductive step, we will consider the last rule (from the Table 1.3) applied in the derivation, supposing correctness of all previous fragments (or subtrees) of the proof.

Case $(\wedge_i)$. For a derivation finishing in an application of this rule, the last step of the proof gives as conclusion φ that should be of the form $(\psi \wedge \phi)$, for formulas ψ and ϕ, that are the premises of the last step of the proof. This is depicted in the following figure.



The left premise is the root of a derivation tree for the sequent $\Gamma \vdash \psi$ and the right one, for the sequent $\Gamma \vdash \phi$. In fact, not all assumptions in Γ need to be open leaves of these subtrees. By induction hypothesis, one has both $\Gamma \models \psi$ and $\Gamma \models \phi$. Thus,

for all assignments that made the formulas in Γ true, the formulas ψ and ϕ are also true, which implies that $(\psi \wedge \phi)$ is true too. Consequently, $\Gamma \models \varphi$.

Case $(\wedge_e)$. For a derivation finishing in an application of this rule, one obtains as conclusion the formula φ from a premise of the form $(\varphi \wedge \psi)$. This is depicted in the figure below.

$$\dfrac{\gamma_1 \quad \cdots \quad \gamma_n}{(\varphi \wedge \psi)}\ \dfrac{}{\varphi}\ (\wedge_e)$$

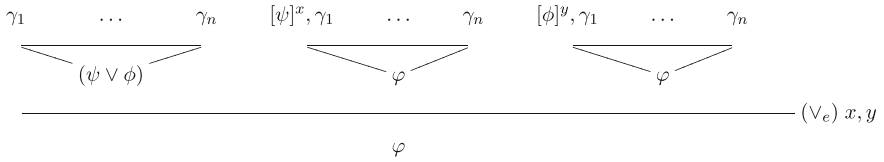
The subtree rooted by the formula $(\varphi \wedge \psi)$ has open leaves labeled with assumptions of the set Γ; not necessarily all these formulas. This subtree is a derivation for the sequent $\Gamma \vdash (\varphi \wedge \psi)$. By induction hypothesis, one has that $\Gamma \models (\varphi \wedge \psi)$, which means that all assignments which make true all formulas in Γ, make also true the formula $(\varphi \wedge \psi)$ and consequently both formulas φ and ψ. Thus, one can conclude that all assignments that make true all formulas in Γ, make also true φ; that is, $\Gamma \models \varphi$.

Case $(\vee_i)$. For a derivation finishing in an application of this rule, the conclusion, that is the formula φ, should be of the form $\varphi = (\psi \vee \phi)$, and the premise of the last rule is ψ as depicted in the following figure.

$$\dfrac{\gamma_1 \quad \cdots \quad \gamma_n}{\psi}\ \dfrac{}{(\psi \vee \phi)}\ (\vee_i)$$

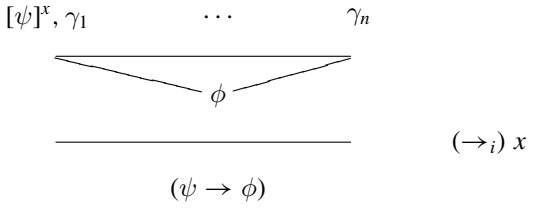
The subtree rooted by the formula ψ and with open leaves labeled by formulas of Γ, corresponds to a derivation for the sequent $\Gamma \vdash \psi$, that by induction hypothesis implies $\Gamma \models \psi$. This implies that all assignments that make the formulas in Γ true, make also ψ true and consequently, the formula $(\psi \vee \phi)$ is true too, under these assignments. Thus, $\Gamma \models \varphi$.

Case $(\vee_e)$. For a derivation of the sequent $\Gamma \vdash \varphi$ that finishes in an application of this rule, one has as premises formulas $(\psi \vee \phi)$, and two repetitions of φ. The former premise labels a root of a subtree with open leaves labeled by assumptions in Γ, that corresponds to a derivation for the sequent $\Gamma \vdash (\psi \vee \phi)$, for some formulas ψ and ϕ. The latter two repetitions of φ, are labeling subtrees with open leaves in Γ and $[\psi]^x$, the first one, and $[\phi]^y$, the second one, as depicted in the figure below.



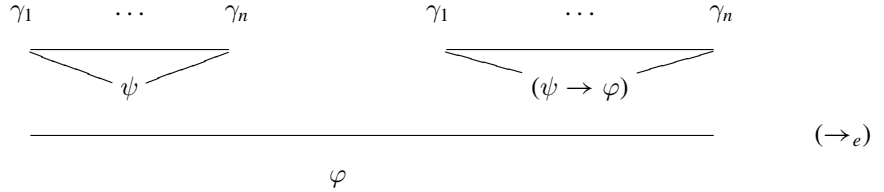
The left subtree whose root is labeled with formula φ, corresponds to a derivation for the sequent $\Gamma, \psi \vdash \varphi$, and the right subtree with φ as root, to a derivation for the sequent $\Gamma, \phi \vdash \varphi$. By induction hypothesis, one has $\Gamma \models (\psi \vee \phi)$, $\Gamma, \psi \models \varphi$ and $\Gamma, \phi \models \varphi$. The first means, that for all assignments that make the formulas in Γ true, $(\psi \vee \phi)$ is also true. And by the semantics of the logical connective $\vee$, $(\psi \vee \phi)$ is true if at least one of the formulas ψ or ϕ is true. In the case that ψ is true, since $\Gamma, \psi \models \varphi$, φ should be true too; in the case in which ϕ is true, since $\Gamma, \phi \models \varphi$, φ should be true as well. Then whenever all formulas in Γ are true, φ is true as well, which implies that $\Gamma \models \varphi$.

Case $(\rightarrow_i)$. For a derivation that finishes in an application of this rule, φ should be of the form $(\psi \rightarrow \phi)$, for some formulas ψ and ϕ. The premise of the last step in this derivation should be the formula ϕ. This formula labels the root of subtree that is a derivation for the sequent $\Gamma, \psi \vdash \phi$. See the next figure.



By induction hypothesis, one has that $\Gamma, \psi \models \phi$, which means that for all assignments that make all formulas in Γ and ψ true, ϕ is also true. Suppose, one has an assignment d, that makes all formulas in Γ true. If ψ is true under this assignment, ϕ is also true. If ψ is false under this assignment, by the semantical interpretation of the connective $\rightarrow$, $\psi \rightarrow \phi$ is also true under this assignment. Thus, one can conclude that for any assignment that makes all formulas in Γ true, the formula φ, that is $\psi \rightarrow \phi$ is true too. Consequently, $\Gamma \models \varphi$.

Case $(\rightarrow_e)$. If the last step of the derivation is $(\rightarrow_e)$, then its premises are formulas of the form ψ and $(\psi \rightarrow \varphi)$, for some formula ψ, as illustrated in the figure below.



The left subtree corresponds to a derivation for the sequent $\Gamma \vdash \psi$ and the right one to a derivation for the sequent $\Gamma \vdash (\psi \to \varphi)$. By induction hypothesis, one has both $\Gamma \models \psi$ and $\Gamma \models (\psi \to \varphi)$. This means that any assignment that makes all formulas in Γ true also makes ψ and $(\psi \to \varphi)$ true. By the semantical interpretation of implication, whenever both ψ and $\psi \to \varphi$ are true, φ should be also true, which implies that $\Gamma \models \varphi$.

Case $(\neg_i)$. When this is the last applied rule in the derivation, φ is of the form $(\neg\psi)$, and the premise of the last step is $\bot$ as depicted in the next figure.

$$\dfrac{\begin{array}{c}[\psi]^x, \gamma_1 \quad \cdots \quad \gamma_n \\ \hline \bot \end{array}}{(\neg\psi)}\ (\neg_i)\ x$$

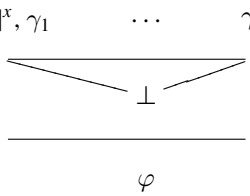
The subtree rooted by $\bot$ has open leaves labeled by formulas in Γ and ψ and corresponds to a proof of the sequent $\Gamma, \psi \vdash \bot$. By induction hypothesis, one has that $\Gamma, \psi \models \bot$, which means that for all assignments that make all formulas in Γ and ψ true, $\bot$ should also be true. But, by the semantical interpretation of $\bot$, this is always false. Then, there is no possible assignment, that makes all formulas in Γ and ψ true. Consequently, any assignment that makes all formulas in Γ true should make ψ false and, by the interpretation of the connective $\neg$, it makes $(\neg\psi)$ true. Thus, one can conclude that $\Gamma \models \varphi$.

Case $(\neg_e)$. For a derivation with last applied rule $(\neg_e)$, the conclusion, that is φ, is equal to the atomic formula $\bot$ and the premises of the last applied rule are formulas ψ and $(\neg\psi)$, for some formula ψ as illustrated in the next figure.

$$\dfrac{\begin{array}{cc}\gamma_1 \quad \cdots \quad \gamma_n & \gamma_1 \quad \cdots \quad \gamma_n \\ \hline \psi & (\neg\psi) \end{array}}{\bot}\ (\neg_e)$$

The derivation has open leaves labeled by formulas in Γ. The left and right subtrees, respectively, rooted by ψ and $\neg\psi$ correspond to derivations for the sequents $\Gamma \vdash \psi$ and $\Gamma \vdash \neg\psi$. By induction hypothesis, one has both $\Gamma \models \psi$ and $\Gamma \models \neg\psi$, which means that any assignment that makes all formulas in Γ true makes also both ψ and $\neg\psi$ true. Consequently, there is no assignment that makes all formulas in Γ true. Thus, one concludes that $\Gamma \models \bot$.

Case (PBC). For a derivation with last applied rule (PBC), the situation is illustrated in the next figure.



(PBC) x

One has $\neg\varphi, \Gamma \vdash \bot$ and by induction hypothesis, $\neg\varphi, \Gamma \models \bot$. The latter implies that no assignment makes $\neg\varphi$ and all formulas in Γ true. Consequently, for any assignment that makes all formulas in Γ true, $\neg\varphi$ should be false and consequently φ true. Thus, one concludes $\Gamma \models \varphi$. □

1.6.2 Completeness of the Propositional Logic

Now, we will prove that the propositional calculus, as given by the rules of natural deduction presented in Table 1.3, is also complete; that is, each logical consequence can be effectively proved through application of rules of the propositional logic. As a preliminary result, we will prove that each valid formula is in fact a formally provable theorem: $\models \varphi$ implies $\vdash \varphi$. Then, we will prove that this holds in general: whenever $\Gamma \models \varphi$, there exists a deduction for the sequent $\Gamma \vdash \varphi$, being Γ a finite set of propositional formulas.

To prove that validity implies provability, an auxiliary lemma is necessary.

Lemma 3 (Truth-values, assignments, and deductions) *Let V be a set of propositional variables, φ be a propositional formula containing only the propositional variables $v_1, \ldots, v_n$ in V and let d be an assignment. Additionally, let $\widehat{v}^d$ denote the formula v whenever $d(v) = T$ and the formula $\neg v$, whenever $d(v) = F$, for $v \in V$. Then, one has*

- *If φ is true under assignment d, then*

$$\widehat{v_1}^d, \ldots, \widehat{v_n}^d \vdash \varphi$$

- *Otherwise,*

$$\widehat{v_1}^d, \ldots, \widehat{v_n}^d \vdash \neg\varphi$$

Proof The proof is by induction on the structure of φ.
IB. The three possible cases are easily verified:
Case $\bot$ for $\varphi = \bot, \vdash \neg\bot$;
Case $\top$ for $\varphi = \top, \vdash \top$;
Case variable for $\varphi = v \in V$, since φ contains only variables in $v_1, \ldots, v_n$, then $\varphi = v_i$, for some $1 \leq i \leq n$. Two possibilities should be considered: if $d(v_i) = T$, one has $\widehat{v_i}^d \vdash v_i$, that is, $v_i \vdash v_i$; if $d(v_i) = F$, one has $\widehat{v_i}^d \vdash \neg v_i$, that is, $\neg v_i \vdash \neg v_i$.
IS. The analysis proceeds by cases according to the structure of φ.

Case $\varphi = (\neg\psi)$. Observe that the set of variables occurring in φ and ψ is the same. In addition, by the semantics of negation, when φ is true under assignment d, ψ should be false and ψ is true under assignment d only if φ is false under this assignment.

By induction hypothesis, whenever ψ is false under assignment d it holds that

$$\widehat{v_1}^d, \ldots, \widehat{v_n}^d \vdash (\neg\psi) = \varphi$$

that is what we need to prove in this case in which φ is true. Also, by induction hypothesis, whenever ψ is true under assignment d one has

$$\widehat{v_1}^d, \ldots, \widehat{v_n}^d \vdash \psi$$

which means that there is a deduction of ψ from the formulas $\widehat{v_1}^d, \ldots, \widehat{v_n}^d$. Thus, also a proof from this set of formulas of $\neg\varphi$ is obtained as below.

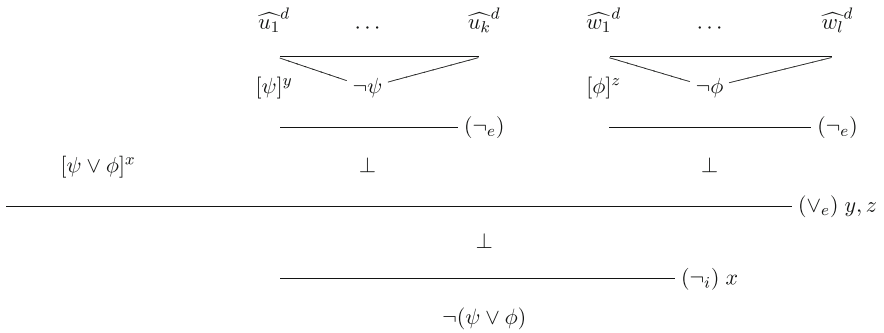
$$\dfrac{\dfrac{\widehat{v_1}^d \quad \cdots \quad \widehat{v_n}^d}{\psi}}{(\neg\neg\psi)}\,(\neg\neg_i)$$

For the cases in which φ is a conjunction, disjunction or implication, of formulas ψ and ϕ, we will use the following notational convention: $\{u_1, \ldots, u_k\}$ and $\{w_1, \ldots, w_l\}$ are the sets of variables occurring in the formulas ψ and ϕ. Observe that these sets are not necessarily disjoint and that their union will give the set of variables $\{v_1, \ldots, v_n\}$ occurring in φ.

Case $\varphi = (\psi \vee \phi)$. On the one side, suppose, φ is false under assignment d. Then, by the semantics of disjunction, both ψ and ϕ are false too, and by induction hypothesis, there are proofs for the sequents

$$\widehat{u_1}^d, \ldots, \widehat{u_k}^d \vdash \neg\psi \quad \text{and} \quad \widehat{w_1}^d, \ldots, \widehat{w_l}^d \vdash \neg\phi$$

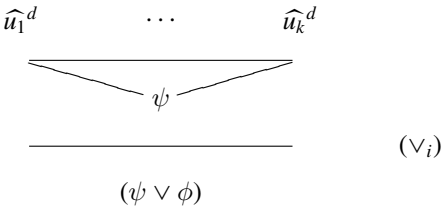
Thus, a proof of $\neg\varphi$, that is $\neg(\psi \vee \phi)$, is obtained combining proofs for these sequents as follows.



On the other side, suppose that φ is true. Then, by the semantics of disjunction, either ψ or ϕ should be true under assignment d (both formulas can be true too). Suppose ψ is true, then by induction hypothesis, we have a derivation for the sequent

$$\widehat{u_1}^d, \ldots, \widehat{u_k}^d \vdash \psi$$

Using this proof we can obtain a proof of the sequent $\widehat{u_1}^d, \ldots, \widehat{u_k}^d \vdash \varphi$, which implies that the desired sequent also holds: $\widehat{v_1}^d, \ldots, \widehat{v_n}^d \vdash \varphi$. The proof is depicted below.



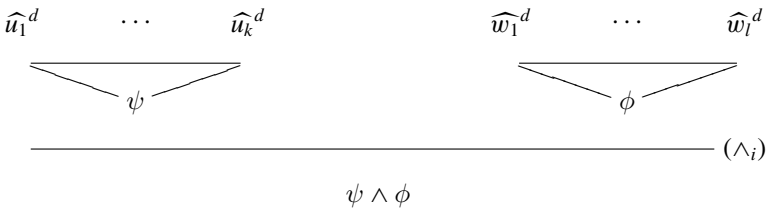
The case in which ψ is false and ϕ is true is done in the same manner, adding an application of rule $(\vee_i)$ at the root of the derivation for the sequent

$$\widehat{w_1}^d, \ldots, \widehat{w_l}^d \vdash \phi$$

Case $\varphi = (\psi \wedge \phi)$. On the one side, suppose, φ is true under assignment d. Then, by the semantics of disjunction, both ψ and ϕ are true too, and by induction hypothesis, there are proofs for the sequents

$$\widehat{u_1}^d, \ldots, \widehat{u_k}^d \vdash \psi \text{ and } \widehat{w_1}^d, \ldots, \widehat{w_l}^d \vdash \phi$$

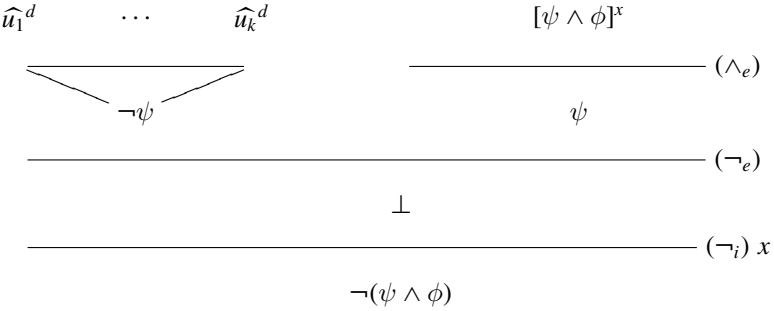
Thus, a proof of φ, that is $(\psi \wedge \phi)$, is obtained combining proofs for these sequents as follows.



On the other side, suppose that φ is false under assignment d. Then, some of the formulas ψ or ϕ should be false, by the semantical interpretation of conjunction. Suppose that ψ is false. The case in which ϕ is false is analogous. Then, by induction hypothesis, one has a derivation for the sequent

$$\widehat{u_1}^d, \ldots, \widehat{u_k}^d \vdash \neg\psi$$

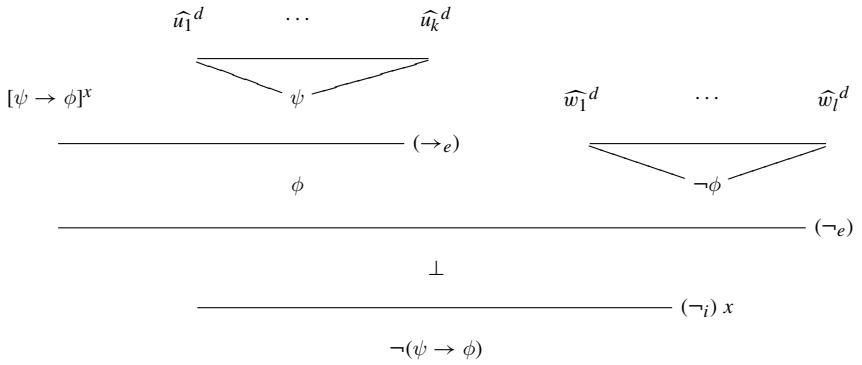
and the derivation for $\neg(\psi \wedge \phi)$, that is for φ, is obtained as depicted below.



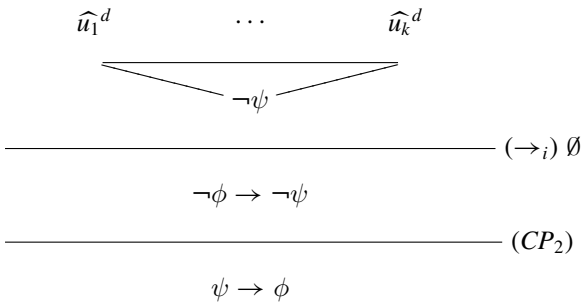
Case $\varphi = (\psi \rightarrow \phi)$. On the one side, suppose, φ is false under assignment d. Then, by the semantics of implication, ψ is true and ϕ false, and by induction hypothesis, there are proofs for the sequents

$$\widehat{u_1}^d, \ldots, \widehat{u_k}^d \vdash \psi \quad \text{and} \quad \widehat{w_1}^d, \ldots, \widehat{w_l}^d \vdash \neg\phi$$

Thus, a proof of $\neg\varphi$, that is $\neg(\psi \rightarrow \phi)$, is obtained combining proofs for these sequents as follows.



On the other side, if φ is true under assignment d, two cases should be considered according to the semantics of implication. First, if ϕ is true, a proof can be obtained from the one for the sequent $\widehat{w_1}^d, \ldots, \widehat{w_l}^d \vdash \phi$, adding an application of rule $(\to_i)$ discharging an empty set of assumptions for ψ and concluding $\psi \to \phi$. Second, if ψ is false, a derivation can be built from the proof for the sequent $\widehat{u_1}^d, \ldots, \widehat{u_k}^d \vdash \neg\psi$ as depicted below.



□

Corollary 1 (Validity and provability for propositional formulas without variables) *Suppose* $\models \varphi$*, for a formula* φ *without occurrences of variables. Then,* $\vdash \varphi$.

Exercise 19 Prove the previous corollary.

Theorem 4 (Completeness: validity implies provability) *For all formula of the propositional logic*

$$\models \varphi \textit{ implies } \vdash \varphi$$

Proof (Sketch) The proof is by an inductive argument on the variables occurring in φ: in each step of the inductive analysis we will get rid of the assumptions in the derivations of φ (built accordingly to Lemma 3) related with one variable of the initial set. Thus, the induction is specifically in the number of variables in φ minus the

number of variables that are been eliminated from the assumptions until the current step of the process. In the end, a derivation for $\vdash \varphi$ without any assumption will be reached.

Suppose one has n variables occurring in φ, say $\{v_1, \ldots, v_n\}$. By the construction of the previous lemma, since $\models \varphi$, one has proofs for all of the 2^n possible designations for the n variables. Selecting a variable v_n one will have 2^{n-1} different proofs of φ with assumption v_n and other 2^{n-1} different proofs with assumption $\neg v_n$. Assembling these proofs with applications of (LEM) (for all formulas $v_i \vee \neg v_i$, for $i \neq n$) and rule $(\vee_e)$, as illustrated below, one obtains a derivation for $v_n \vdash \varphi$ and $\neg v_n \vdash \varphi$, from which a proof for $\vdash \varphi$ is also obtained using (LEM) (for $v_n \vee \neg v_n$) and $(\vee_e)$. The inductive sketch of the proof is as follows.

IB. The case in which φ has no occurrences of variables holds by the Corollary 1. Consider φ has only one variable v_1, Then by a simple application of rule $(\vee_e)$, proofs for $v_1 \vdash \varphi$ and $\neg v_1 \vdash \varphi$, are assembled as below obtaining a derivation for $\vdash \varphi$. The existence of proofs for $v_1 \vdash \varphi$ and $\neg v_1 \vdash \varphi$ is guaranteed by Lemma 3.

$$\dfrac{(v_1 \vee \neg v_1) \qquad \begin{array}{c}[v_1]^x \\ \vdots \\ \varphi\end{array} \qquad \begin{array}{c}[\neg v_1]^y \\ \vdots \\ \varphi\end{array}}{\varphi}\,(\vee_e)\ x, y$$

IS. Suppose φ has $n > 1$ variables. Since $\models \varphi$, by Lemma 3 one has 2^{n-1} different derivations for $v_n, \widehat{v_1}^d, \ldots, \widehat{v_{n-1}}^d \vdash \varphi$ as well for $\neg v_n, \widehat{v_1}^d, \ldots, \widehat{v_n}^d \vdash \varphi$, for all possible designations d. To get rid of the variable v_n one can use these derivations and (LEM) as below.

$$\dfrac{(v_n \vee \neg v_n) \qquad \begin{array}{c}[\widehat{v_1}^d] \ldots [\widehat{v_{n-1}}^d][v_n]^x \\ \vdots \\ \varphi\end{array} \qquad \begin{array}{c}[\widehat{v_1}^d] \ldots [\widehat{v_{n-1}}^d][\neg v_n]^y \\ \vdots \\ \varphi\end{array}}{\varphi}\,(\vee_e)\ x, y$$

In this manner, one builds, for each variable assignment d, a derivation for $\widehat{v_1}^d, \ldots, \widehat{v_{n-1}}^d \vdash \varphi$. Proceeding in this way, that is using (LEM) for other variables and assembling the proofs using the rule $(\vee_e)$ one will be able to get rid of all other variables until a derivation for $\vdash \varphi$ is obtained.

To let things clearer to the reader, notice that the first step analyzed above implies that there are derivations ∇ and ∇', respectively, for the sequents $\widehat{v_1}^d, \ldots, \widehat{v_{n-2}}^d$, $v_{n-1} \vdash \varphi$ and $\widehat{v_1}^d, \ldots, \widehat{v_{n-2}}^d, \neg v_{n-1} \vdash \varphi$. This is possible since in the previous analysis the assignment d is arbitrary; then, derivations as the one depicted above exist for assignments that map v_n either to true or false. Thus, a derivation for

$\widehat{v_1}^{d}, \ldots, \widehat{v_{n-2}}^{d} \vdash \varphi$ is obtained using (LEM) for the formula $v_{n-1} \vee \neg v_{n-1}$, the derivations ∇ and ∇', and the rule $(\vee_e)$, that will discharge the assumptions $[v_{n-1}]$ and $[\neg v_{n-1}]$ in the derivations ∇ and ∇', respectively. □

Remark 3 To clarify the way in which derivations are assembled in the previous inductive proof, let us consider the case of a valid formula φ with three propositional variables p, q, and r and for brevity let $\nabla_{000}, \nabla_{001}, \ldots, \nabla_{111}$, denote derivations for $p, q, r \vdash \varphi$; $p, q, \neg r \vdash \varphi$; $\ldots$, $\neg p, \neg q, \neg r \vdash \varphi$, respectively. Notice that the existence of derivations ∇_{ijk}, for $i, j, k = \{0, 1\}$ is guaranteed by Lemma 3.

Derivations, ∇_{00} for $p, q \vdash \varphi$ and ∇_{01} for $p, \neg q \vdash \varphi$ are obtained as illustrated below.

$$\nabla_{00}: \quad \dfrac{r \vee \neg r \qquad \begin{matrix}[p]^x[q]^y[r]^z \\ \nabla_{000} \\ \varphi\end{matrix} \qquad \begin{matrix}[p]^x[q]^y[\neg r]^{z'} \\ \nabla_{001} \\ \varphi\end{matrix}}{\varphi}\,(\vee_e)\ z, z'$$

$$\nabla_{01}: \quad \dfrac{r \vee \neg r \qquad \begin{matrix}[p]^x[\neg q]^{y'}[r]^z \\ \nabla_{010} \\ \varphi\end{matrix} \qquad \begin{matrix}[p]^x[\neg q]^{y'}[\neg r]^{z'} \\ \nabla_{011} \\ \varphi\end{matrix}}{\varphi}\,(\vee_e)\ z, z'$$

Combining the two previous derivations, a proof ∇_0 is obtained for $p \vdash \varphi$ as follows.

$$\nabla_0: \quad \dfrac{q \vee \neg q \qquad \begin{matrix}[p]^x[q]^y \\ \nabla_{00} \\ \varphi\end{matrix} \qquad \begin{matrix}[p]^x[\neg q]^{y'} \\ \nabla_{01} \\ \varphi\end{matrix}}{\varphi}\,(\vee_e)\ y, y'$$

Analogously, combining proofs ∇_{100} and ∇_{101} one obtains derivations ∇_{10} and ∇_{11} respectively for $\neg p, q \vdash \varphi$ and $\neg p, \neg q \vdash \varphi$. From These two derivations it's possible to build a derivation ∇_1 for $\neg p \vdash \varphi$. Finally, from ∇_0 and ∇_1, proofs for $p \vdash \varphi$ and $\neg p \vdash \varphi$, one obtains the desired derivation for $\vdash \varphi$.

The whole assemble, that is a derivation ∇ for $\vdash \varphi$, is depicted below. Notice the drawback of being exponential in the number of variables occurring in the valid formula φ.

$$\dfrac{\dfrac{\dfrac{\nabla_{000}\nabla_{001}}{\varphi}\quad\dfrac{\nabla_{010}\nabla_{011}}{\varphi}}{\varphi}\quad\dfrac{\dfrac{\nabla_{100}\nabla_{101}}{\varphi}\quad\dfrac{\nabla_{110}\nabla_{111}}{\varphi}}{\varphi}}{\varphi}$$

Exercise 20 Build a derivation for the instance of Peirce's law in propositional variables p and q according to the inductive construction of the proof of the completeness (Theorem 4). That is, first build derivations for $p, q \vdash ((p \to q) \to p) \to p$, $p, \neg q \vdash ((p \to q) \to p) \to p$, $\neg p, q \vdash ((p \to q) \to p) \to p$ and $\neg p, \neg q \vdash ((p \to q) \to p) \to p$, and then assemble these proofs to obtain a derivation for $\vdash ((p \to q) \to p) \to p$.

Finally, we proceed to prove the general version of the completeness of propositional logic, that is

$$\Gamma \models \varphi \quad \text{implies} \quad \Gamma \vdash \varphi$$

Theorem 5 (Completeness of Propositional Logic) *Let Γ be a finite set of propositional formulas, and φ be a propositional formula. If $\Gamma \models \varphi$ then $\Gamma \vdash \varphi$.*

Proof Let $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$. Initially, notice that

$$\gamma_1, \ldots, \gamma_n \models \varphi \quad \text{implies} \quad \models \gamma_1 \to (\gamma_2 \to (\cdots(\gamma_n \to \varphi)\cdots))$$

Indeed, by contraposition, $\gamma_1 \to (\gamma_2 \to (\cdots(\gamma_n \to \varphi)\cdots))$ can only be false if all formulas γ_i, for $i = 1, \ldots, n$ are true and φ is false, which gives a contradiction to the assumption that φ is a logical consequence of Γ.

By, Theorem 4, the valid formula $\gamma_1 \to (\gamma_2 \to (\cdots(\gamma_n \to \varphi)\cdots))$ should be provable, that is, there exists a derivation, say ∇, for

$$\vdash \gamma_1 \to (\gamma_2 \to (\cdots(\gamma_n \to \varphi)\cdots))$$

To conclude, a derivation ∇' for $\gamma_1, \ldots, \gamma_n \vdash \varphi$ can be built from the derivation ∇ by assuming $[\gamma_1]$, $[\gamma_2]$, etc. and eliminating the premises of the implication γ_1, γ_2, etc. by repeatedly applications the rule $(\to_e)$, as depicted below.

$$\dfrac{\dfrac{\dfrac{\nabla}{\gamma_1 \to (\gamma_2 \to (\cdots(\gamma_n \to \varphi)\cdots))}\quad [\gamma_1]^{u_1}}{\gamma_2 \to (\cdots(\gamma_n \to \varphi)\cdots)}(\to_e)\quad [\gamma_2]^{u_2}}{\vdots}(\to_e)$$
$$\dfrac{[\gamma_n]^{u_n}\qquad \gamma_n \to \varphi}{\varphi}(\to_e)$$

□

Additional Exercise 21 As explained before, the classical propositional logic can be characterized by any of the equivalent rules (PBC), $(\neg\neg_e)$ or (LEM). Show that Peirce's law is also equivalent to any of these rules. In other words, build intuitionistic proofs for the rules (PBC), $(\neg\neg_e)$ and (LEM) assuming the rule:

$$\frac{}{((\phi \to \psi) \to \phi) \to \phi}\ (\text{LP})$$

Next, prove (LP) in three different ways: each proof should be done in the intuitionistic logic assuming just one of (PBC), $(\neg\neg_e)$ and (LEM) at a time.

Additional Exercise 22 Prove the following sequents:

a. $\phi \to (\psi \to \gamma), \phi \to \psi \vdash \phi \to \gamma$
b. $(\phi \vee (\psi \to \phi)) \wedge \psi \vdash \phi$
c. $\phi \to \psi \vdash ((\phi \wedge \psi) \to \phi) \wedge (\phi \to (\phi \wedge \psi))$
d. $\vdash \psi \to (\phi \to (\phi \to (\psi \to \phi)))$

Chapter 2
Derivations and Proofs in the Predicate Logic

2.1 Motivation

The propositional logic has several limitations for expressing ideas; mainly, it is not possible to quantify over sets of individuals and reason about them. These limitations can be better explained through examples

"Every prime number bigger than 2 is odd"
"There exists a prime number greater than any given natural number"

In the language of the propositional logic, this kind of properties can only be represented by a propositional variable because there is no way to split this information into simpler propositions joined by connectives and able to express the quantification over the natural numbers. In fact, the information in these sentences includes observations about sets of prime numbers, odd numbers, natural numbers, and quantification over them, and these relations cannot be straightforwardly captured in the language of propositional logic.

In order to overcome these limitations of the expressive power of the propositional logic, we extend its language with variables which range over individuals, and quantification over these variables. Thus, in this chapter we present the *predicate logic*, also known as *first-order logic*. In order to obtain a language with abilities to identify the required additional information, we need to extend the propositional language and provide a more expressive deductive calculus.

2.2 Syntax of the Predicate Logic

The language of the first-order predicate logic has two kinds of expressions: *terms* and *formulas*. While in the language of propositional logic formulas that are built up from propositional variables, in the predicate logic they are built from atomic formulas, that are relational formulas expressing properties of terms such as "prime(2)", "prime(x)",

M. Ayala-Rincón and F.L.C. de Moura, *Applied Logic for Computer Scientists*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-51653-0_2

"x is bigger than 2", etc. Formulas are built from relational formulas using the logical connectives as in the case of propositional logic, but in predicate logic also quantifiers over variables will be possible. Terms and basic relational formulas are built out of variables and two sets of symbols $\mathbb{F}$ and $\mathbb{P}$. Each function symbol in $\mathbb{F}$ and each predicate symbol in $\mathbb{P}$ come with its fixed arity (that is, the number of its arguments). *Constants* can be seen as function symbols of arity zero. No predicate symbols with arity zero are allowed. This is the part of the language that is flexible since the sets $\mathbb{F}$ and $\mathbb{P}$ can be chosen arbitrarily.

Intuitively, predicates are functions that represent properties of terms. In order to define predicate formulas, we first define terms, and to do so, we assume an enumerable set $\mathbb{V}$ of *term variables*.

Definition 13 (*Terms*) A *term* t is defined inductively as follows:

1. Any variable $x \in \mathbb{V}$ is a term;
2. If $t_1, t_2, \ldots, t_n$ are terms, and $f \in \mathbb{F}$ is a function symbol with arity $n \geq 0$ then $f(t_1, t_2, \ldots, t_n)$ is a term. A function of arity zero is a constant.

Notation 1 We follow the usual notational convention for terms. Constant symbols, function symbols, arbitrary terms, and variables are denoted by Roman lower case letters, respectively, of the first, second, third, and fourth quarters of the alphabet: $a, b, \ldots$, for constant symbols; $f, g, \ldots$, for function symbols; $s, t, \ldots$ for arbitrary terms and; x, y, z, for variables.

Terms, as given in the previous definition, could be equivalently presented by the following syntax:

$$t ::= x \;\; || \;\; f(t, \ldots, t)$$

Definition 14 (*Variable occurrence*) The set of variables occurring in a term t, denoted by $\mathtt{var}(t)$, is inductively defined as follows:

- If $t = x$ then $\mathtt{var}(t) = \{x\}$
- If $t = f(t_1, \ldots, t_n)$ then $\mathtt{var}(t) = \mathtt{var}(t_1) \cup \cdots \cup \mathtt{var}(t_n)$

We define the substitution of the term u for x in the term t, written $t[x/u]$, as the replacement of all occurrences of x in t by u. Formally, we have the following definition.

Definition 15 (*Term Substitution*) Let t, u be terms, and x, a variable. We define $t[x/u]$ inductively as follows:

- $x[x/u] = u$;
- $y[x/u] = y$, for $y \neq x$;
- $f(t_1, \ldots, t_n)[x/u] = f(t_1[x/u], \ldots, t_n[x/u])$ $(n \geq 0)$.

Now we are a ready to define the formulas of the predicate logic:

Definition 16 (*Formulas*) The set of formulas of the first-order predicate logic over a variable set $\mathbb{V}$ and a symbol set $S = (\mathbb{F}, \mathbb{P})$ is inductively defined as follows:

1. $\bot$ and $\top$ are formulas;
2. If $p \in \mathbb{P}$ with arity $n > 0$, and $t_1, t_2, \ldots, t_n$ are terms then $p(t_1, t_2, \ldots, t_n)$ is a formula;
3. If φ is a formula then so is $(\neg\varphi)$;
4. If φ_1 and φ_2 are formulas then so are $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$ and $(\varphi_1 \rightarrow \varphi_2)$;
5. If $x \in \mathbb{V}$ and φ is a formula then $(\forall_x \varphi)$ and $(\exists_x \varphi)$ are formulas.

The symbol $\forall_x$ (resp. $\exists_x$) means "for all x" (resp. "there exists a x"), and the formula φ is the *body* of the formula $(\forall_x \varphi)$ (resp. $(\exists_x \varphi)$). Since quantification is restricted to variable terms, the defined language corresponds to a so-called *first-order language*.

The set of formulas of the predicate logic have the following syntax:

$$\varphi ::= p(t, \ldots, t) \;||\; \bot \;||\; \top \;||\; (\neg\varphi) \;||\; (\varphi \wedge \varphi) \;||\; (\varphi \vee \varphi) \;||\; (\varphi \rightarrow \varphi) \;||\; (\forall_x \varphi) \;||\; (\exists_x \varphi)$$

Formulas of the form $p(t_1, \ldots, t_n)$ are called *atomic formulas* because they cannot be decomposed into simpler formulas. As usual, parenthesis are used to avoid ambiguities and the external ones will be omitted. The quantifiers $\forall_x$ and $\exists_x$ bind the variable x in the body of the formula. This idea is formalized by the notion of *scope of a quantifier*:

Definition 17 (*Scope of quantifiers, free and bound variables*) The scope of $\forall_x$ (resp. $\exists_x$) in the formula $\forall_x \varphi$ (resp. $\exists_x \varphi$) is the body of the quantified formula: φ. An occurrence of a variable x in the scope of $\forall_x$ or $\exists_x$ is called *bound*. An occurrence of a variable that is not bound is called *free*.

Since the body of a quantified formula can have occurrences of other quantified formulas that abstract the same variable symbol, it is necessary to provide more precise mechanisms to build the sets of free and bound variables of a predicate formula. This can be done inductively according to the following definitions:

Definition 18 (*Construction of the set of free variable*) Let φ be a formula of the predicate logic. The set of free variables of φ, denoted by $\mathtt{fv}(\varphi)$, is inductively defined as follows:

1. $\mathtt{fv}(\bot) = \mathtt{fv}(\top) = \emptyset$;
2. $\mathtt{fv}(p(t_1, \ldots, t_n)) = \mathtt{var}(t_1) \cup \ldots \cup \mathtt{var}(t_n)$;
3. $\mathtt{fv}(\neg\varphi) = \mathtt{fv}(\varphi)$;
4. $\mathtt{fv}(\varphi \Box \psi) = \mathtt{fv}(\varphi) \cup \mathtt{fv}(\psi)$, where $\Box \in \{\wedge, \vee, \rightarrow\}$;
5. $\mathtt{fv}(Q_x \varphi) = \mathtt{fv}(\varphi) \setminus \{x\}$, where $Q \in \{\forall, \exists\}$.

A formula without occurrences of free variables is called a *sentence*.

Definition 19 (*Construction of the set of bound variables*) Let φ be a formula of the predicate logic. The set of bound variables of φ, denoted by $\mathtt{bv}(\varphi)$, is inductively defined as follows:

1. $\mathtt{bv}(\bot) = \mathtt{bv}(\top) = \emptyset$;
2. $\mathtt{bv}(p(t_1, \ldots, t_n)) = \emptyset$;
3. $\mathtt{bv}(\neg\varphi) = \mathtt{bv}(\varphi)$;
4. $\mathtt{bv}(\varphi \Box \psi) = \mathtt{bv}(\varphi) \cup \mathtt{bv}(\psi)$, where $\Box \in \{\wedge, \vee, \rightarrow\}$;
5. $\mathtt{bv}(Q_x\varphi) = \mathtt{bv}(\varphi) \cup \{x\}$, where $Q \in \{\forall, \exists\}$.

Informally, the name of a bound variable is not important in the sense that it can be *renamed* to any *fresh* name without changing the semantics of the term. For instance, the formulas $\forall_x(x \leq x)$, $\forall_y(y \leq y)$ and $\forall_z(z \leq z)$ represent the very same object. The sole restriction that needs to be considered is that *variable capture* is forbidden, i.e., no free variable can become bound after a renaming of a variable. For instance, if p denotes a binary predicate then $\forall_x p(x, y)$ is a renaming of $\forall_z p(z, y)$, while $\forall_y p(y, y)$ is not. The next definition will formalize the notion of substitution. The capture of free variables by a substitution is also forbidden, and we assume that a renaming of bound variables is always performed when necessary to avoid capture.

Definition 20 (*Substitution*) Let φ be a formula of the predicate logic. The substitution of x by t in φ, written $\varphi[x/t]$, is inductively defined as follows:

1. $\bot[x/t] = \bot$ and $\top[x/t] = \top$;
2. $p(t_1, \ldots, t_n)[x/t] = p(t_1[x/t], \ldots, t_n[x/t])$;
3. $(\neg\psi)[x/t] = \neg(\psi[x/t])$;
4. $(\psi \Box \gamma)[x/t] = (\psi[x/t]) \Box (\gamma[x/t])$, where $\Box \in \{\wedge, \vee, \rightarrow\}$;
5. $(Q_y\psi)[x/t] = Q_y(\psi[x/t])$, where $Q \in \{\exists, \forall\}$, and renaming of bound variables is assumed to avoid capture of variables.

Example 6 Consider the following applications of substitution:

- $(\forall_x p(y))[y/x] = \forall_z p(y)[y/x] = \forall_z p(y[y/x]) = \forall_z p(x)$ and
- $(\forall_x p(x))[x/t] = \forall_y p(y)[x/t] = \forall_y p(y[x/t]) = \forall_y p(y)$.

Notice that in the second application, renaming x as y was necessary to avoid capture.

The necessary renamings to avoid capture of variables in substitutions can be implemented in several ways. For instance, it can be done by modifying item *5* in the definition of substitution in such a way that before propagating the substitution inside the scope of a quantified formula of the form $(Q_x\varphi)[x/t]$, where $Q \in \{\forall, \exists\}$, it is checked whether $x = y$ or $x \in \mathtt{fv}(t)$: whenever $x = y$ or $x \in \mathtt{fv}(t)$ renaming the quantified variable name x as a fresh variable name z is applied, in other case no renaming is needed

$$(Q_x\varphi)[y/t] = \begin{cases} (Q_z\varphi[x/z][y/t]), & \text{if } x = y \text{ or } x \in \mathtt{fv}(t), \\ (Q_x\varphi[y/t]), & \text{otherwise.} \end{cases}$$

The size of predicate expressions (terms and formulas) is defined in the usual manner.

Definition 21 (*Size of predicate expressions*) Let t be a predicate term and φ a predicate formula. The size of t, denoted as $|t|$, is recursively defined as follows:

- $|x| = 1$, for $x \in \mathbb{V}$;
- $|f(t_1, \ldots, t_n)| = 1 + |t_1| + \cdots + |t_n|$, for $n \geq 0$.

The size of φ, denoted as $|\varphi|$, is recursively defined as follows:

- $|\bot| = |\top| = 1$;
- $|p(t_1, \ldots, t_n)| = 1 + |t_1| + \cdots + |t_n|$, for $n \geq 1$;
- $|(\neg\psi)| = 1 + |\psi|$;
- $|(\psi \Box \gamma)| = 1 + |\psi| + |\gamma|$, where $\Box \in \{\wedge, \vee, \rightarrow\}$;
- $|(Q_y \psi)| = 1 + |\psi|$, where $Q \in \{\exists, \forall\}$.

Exercise 23

a. Consider a predicate formula φ and a term t. Prove that there are no bound variables in the new occurrences of t in the formula $\varphi[x/t]$. For doing this use induction on the structure of φ. Of course, occurrences of the term t in the original formula φ might be under the scope of quantifiers and consequently variables occurring in these subterms would be bound.
b. Let k be the number of free occurrences of the variable x in the predicate formula φ. Prove, also by induction on φ, that the size of the term $\varphi[x/t]$ is given by $k|t| + |\varphi| - k$.
c. For $x \neq y$, prove also that:
   i. $\varphi[x/s][x/t] = \varphi[x/s[x/t]]$;
   ii. $\varphi[x/s][y/t] = \varphi[x/s[y/t]][y/t]$, if $y \notin \mathtt{var}(t)$;
   iii. $\varphi[x/s][y/t] = \varphi[y/t][x/s]$, if $x \notin \mathtt{var}(t)$ and $y \notin \mathtt{var}(s)$.

2.3 Natural Deduction in the Predicate Logic

The set of rules of natural deduction for the predicate logic is an extension of the set presented for the propositional logic. The rules for conjunction, disjunction, implication, and negation have the same shape, but note that now the formulas are the ones of predicate logic. In this section, we also discuss the minimal, intuitionistic, and classical predicate logic. Thus the rules are those in Table 1.2, without the rule $(\bot_e)$ for the minimal predicate logic and with this rule for the intuitionistic predicate logic, and in Table 1.3 for the classical predicate logic, plus four additional rules for dealing with quantified formulas.

We start by expanding the set of natural deduction rules with the ones for quantification. The first one is the elimination rule for the universal quantifier:

$$\frac{\forall_x \varphi}{\varphi[x/t]} \ (\forall_e)$$

The intuition behind this rule is that from a proof of $\forall_x \varphi$, we can conclude $\varphi[x/t]$, where t is any term. This transformation is done by the substitution operator previously defined that replaces every free occurrence of x by an arbitrary term t in φ. According to the substitution operator, "every" occurrence of x in φ is replaced with the "same" term t. The following example shows an application of $(\forall_e)$ in a derivation:

Example 7 $\forall_x p(a, x), \forall_x \forall_y (p(x, y) \rightarrow p(f(x), y)) \vdash p(f(a), f(a))$.

$$\dfrac{\dfrac{\forall_x p(a, x)}{p(a, f(a))}\,(\forall_e) \qquad \dfrac{\dfrac{\forall_x \forall_y (p(x, y) \rightarrow p(f(x), y))}{\forall_y p(a, y) \rightarrow p(f(a), y)}\,(\forall_e)}{p(a, f(a)) \rightarrow p(f(a), f(a))}\,(\forall_e)}{p(f(a), f(a))}\,(\rightarrow_e)$$

Note that the application of $(\rightarrow_e)$ is identical to what is done in the propositional calculus, except for the fact that now it is applied to predicate formulas.

The introduction rule for the universal quantifier is more subtle. In order to prove $\forall_x \varphi$ one needs first to prove $\varphi[x/x_0]$ in such a way that no open assumption in the derivation of $\varphi[x/x_0]$ can contain occurrences of x_0. This restriction is necessary to guarantee that x_0 is general enough and can be understood as "any" term, i.e., nothing has been assumed concerning x_0. The $(\forall_i)$ rule is given by

$$\dfrac{\varphi[x/x_0]}{\forall_x \varphi}\,(\forall_i)$$

where x_0 is a fresh variable not occurring in any open assumption in the derivation of $\varphi[x/x_0]$.

Example 8 $\forall_x (p(x) \wedge q(x)) \vdash \forall_x (p(x) \rightarrow q(x))$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall_x (p(x) \wedge q(x))}{p(x_0) \wedge q(x_0)}\,(\forall_e)}{q(x_0)}\,(\wedge_e)}{p(x_0) \rightarrow q(x_0)}\,(\rightarrow_i)\emptyset}{\forall_x (p(x) \rightarrow q(x))}\,(\forall_i)$$

Note that the formula $p(x_0) \rightarrow q(x_0)$ depends only on the hypothesis $\forall_x (p(x) \wedge q(x))$, which does not contain x_0. Thus x_0 might be considered arbitrary, which allows the generalization through application of rule $(\forall_i)$. In fact, note that the above proof of $p(x_0) \rightarrow q(x_0)$ could be done for any other term, say t instead x_0, which explains the generality of x_0 in the above example.

The introduction rule for the existential quantifier is as follows:

$$\frac{\varphi[x/t]}{\exists_x \varphi}\ (\exists_i)$$

where t is any term.

Example 9 $\forall_x q(x) \vdash \exists_x q(x)$.

$$\dfrac{\dfrac{\forall_x q(x)}{q(x_0)}\ (\forall_e)}{\exists_x q(x)}\ (\exists_i)$$

Similarly to $(\forall_i)$, the elimination rule for the existential quantifier is more subtle:

$$\dfrac{\exists_x \varphi \qquad \begin{array}{c}[\varphi[x/x_0]]^u \\ \vdots \\ \chi\end{array}}{\chi}\ (\exists_e)\ u$$

This rule requires the variable x_0 be a fresh variable neither occurring in any other open assumption than in $[\varphi[x/x_0]]^u$ itself nor in the conclusion $\mathtt{fv}(\chi)$. The intuition of this rule might be explained as follows: knowing that $\exists_x \varphi$ holds, if assuming that an arbitrary x_0 witnesses the property φ, i.e., assuming $[\varphi[x/x_0]]^u$, one can infer χ, then χ holds in general. This kind of analysis is done, for instance, when properties about numbers are inferred from the knowledge of the existence of prime numbers of arbitrary size, or (good/bad) properties about institutions are inferred from the knowledge of the existence of the (good/bad) qualities of some individuals in their staffs. These general properties are inferred without knowing specific prime numbers or without knowing who are specifically the (good/bad) individuals in the institutions.

Example 10 This example attempts to bring a little bit intuition about the use of these rules. Let p, q, and r be predicate symbols with the intended meanings: $p(z)$ means "z is a planet different from the earth with similar characteristics"; $q(y)$ means "country y adopts action to mitigate global warming" and $r(x, y)$ means "x is a leader, who works in the ministry of agriculture or environment of country y and who is worried about climate change". Thus, from the hypotheses $\forall_y \exists_x r(x, y)$, $\forall_y \forall_x (r(x, y) \to q(y))$ and $\forall_z (\forall_y q(y) \to \neg p(z))$, we can infer that we do not need a "Planet B" as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall_y \exists_x r(x,y)}{\exists_x r(x,c_0)}\,(\forall_e) \quad \dfrac{[r(l_0,c_0)]^u \qquad \dfrac{\dfrac{\forall_y \forall_x (r(x,y) \to q(y))}{\forall_x (r(x,c_0) \to q(c_0))}\,(\forall_e)}{r(l_0,c_0) \to q(c_0)}\,(\to_e)}{q(c_0)}}{q(c_0)}\,(\exists_e)\ u}{\forall_y q(y)}\,(\forall_e) \quad \dfrac{\forall_z (\forall_y q(y) \to \neg p(z))}{\forall_y q(y) \to \neg p(B)}\,(\forall_e)}{\neg p(B)}\,(\to_e)$$

Example 11 The use of substitution in natural deduction rules for quantifiers is illustrated in this example. Initially, consider a unary predicate p. Below, it is depicted a derivation for $\exists_x\, p(x) \vdash \neg\forall_x \neg p(x)$.

$$\dfrac{\exists_x p(x) \qquad \dfrac{\dfrac{[p(x_0)]^u \qquad \dfrac{[\forall_x \neg p(x)]^v}{\neg p(x_0)}\,(\forall_e)}{\bot}\,(\neg_e)}{\neg\forall_x \neg p(x)}\,(\neg_i)\ v}{\neg\forall_x \neg p(x)}\,(\exists_e)\ u$$

Now, consider a predicate formula φ and a variable x that might or might not occur free in φ. The next derivation, denoted as ∇_3, proofs that $\vdash \exists_x \varphi \to \neg\forall_x \neg\varphi$. Despite the proof for φ appears to be the same than the one above for the unary predicate p, several subtle points should be highlighted. In the application of rule $(\exists_e)$ in the derivation ∇_3, it is forbidden the selection of a witness variable "y", to be used in the witness assumption $[\varphi[x/y]]^w$, such that y belongs to the set of free variables occurring in φ. Indeed, y should be a *fresh* variable. To understand this restriction, consider $\varphi = q(y,x)$ and suppose the intended meaning of q is "x is the double of y". If the existential formula is $\exists_x p(y,x)$ the witness assumption cannot be $p(y,x)[x/y] = p(y,y)$, since this selection of "y" is not arbitrary.

$$\dfrac{\dfrac{[\exists_x \varphi]^u \qquad \dfrac{\dfrac{[\varphi[x/y]]^w \qquad \dfrac{[\forall_x \neg\varphi]^v}{\neg\varphi[x/y]}\,(\forall_e)}{\bot}\,(\neg_e)}{\bot}\,(\exists_e)\ w}{\neg\forall_x \neg\varphi}\,(\neg_i)\ v}{\exists_x \varphi \to \neg\forall_x \neg\varphi}\,(\to_i)\ u$$

The rules for quantification discussed so far are summarized in Table 2.1. These rules together with the deduction rules for introduction and elimination of the con-

Table 2.1 Natural deduction rules for quantification

Introduction rules	Elimination rules
$\dfrac{\varphi[x/x_0]}{\forall_x \varphi}\ (\forall_i)$ where x_0 cannot occur free in any open assumption.	$\dfrac{\forall_x \varphi}{\varphi[x/t]}\ (\forall_e)$
$\dfrac{\varphi[x/t]}{\exists_x \varphi}\ (\exists_i)$	$\dfrac{\exists_x \varphi \qquad \begin{array}{c}[\varphi[x/x_0]]^u \\ \vdots \\ \chi\end{array}}{\chi}\ (\exists_e)\ u$ where x_0 cannot occur free in any open assumption on the right and in χ.

nectives: $\wedge$, $\vee$, $\neg$, and $\rightarrow$, conform the set of natural deduction rules for the minimal predicate logic (that is, rules in Tables 2.1 and 1.2 except rule $(\bot_e)$). If in addition, we include the intuitionistic absurdity rule, we obtain the natural deduction calculus for the intuitionistic predicate logic (that is all rules in Tables 2.1 and 1.2). The classical predicate calculus is obtained from the intuitionistic one, changing the intuitionistic absurdity rule by the rule PBC (that is, rules in Tables 2.1 and 1.3).

Example 12 The sequent $\vdash \exists_x \neg\varphi \rightarrow \neg\forall_x \varphi$ has the following intuitionistic proof ∇_1:

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall_x \varphi]^v}{\varphi[x/y]}\ (\forall_e) \qquad [\neg\varphi[x/y]]^w}{\bot}\ (\neg_e)}{\bot}\ (\exists_e)\ w}{\neg\forall_x \varphi}\ (\neg_i)\ v}{\exists_x \neg\varphi \rightarrow \neg\forall_x \varphi}\ (\rightarrow_i)\ u \qquad [\exists_x \neg\varphi]^u$$

The proof ∇_1 can be used to prove the sequent $\vdash \forall_x\varphi \to \neg\exists_x\neg\varphi$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\nabla_1}{\exists_x\neg\varphi \to \neg\forall_x\varphi} \qquad \dfrac{}{[\exists_x\neg\varphi]^v}}{\neg\forall_x\varphi}(\to_e) \qquad \dfrac{}{[\forall_x\varphi]^w}}{\bot}(\neg_e)}{\neg\exists_x\neg\varphi}(\neg_i)\ v}{\forall_x\varphi \to \neg\exists_x\neg\varphi}(\to_i)\ w$$

Exercise 24 Prove intuitionistically that $\neg\exists_x\varphi \dashv\vdash \forall_x\neg\varphi$.

Exercise 25 Prove that:

a. if x does not occur free in ψ then prove that $(\exists_x\phi) \to \psi \vdash \forall_x(\phi \to \psi)$; and
b. if x does not occur free in ψ then prove that $(\forall_x\phi) \to \psi \vdash \exists_x(\phi \to \psi)$.

Exercise 26 Prove that

a. $(\forall_x\phi) \wedge (\forall_x\psi) \dashv\vdash \forall_x(\phi \wedge \psi)$; and
b. $(\exists_x\phi) \vee (\exists_x\psi) \dashv\vdash \exists_x(\phi \vee \psi)$.

Exercise 27 Prove that $\forall_x(p(x) \to \neg q(x)) \vdash \neg(\exists_x(p(x) \wedge q(x)))$.

The interpretation of formulas in the classical logic is different from the one in the intuitionistic logic. While in the intuitionistic logic the goal is to "have a constructive proof" of a formula φ, in the classical logic the goal is to "establish a proof of the truth" of φ. For instance, a classical proof admits the truth of a formula of the form $\exists_x\varphi$ without having an explicit witness for x. Such kind of proof (without an explicit witness for the existential) is not accepted in the intuitionistic logic. As an example, suppose that one wants to prove that there exists two irrational numbers x and y such that x^y is rational. If $r(x)$ means that "x is a rational number" then one aims to prove the sequent $\vdash \exists_x\exists_y(\neg r(x) \wedge \neg r(y) \wedge r(x^y))$. In order to do so, we assume some obvious facts in algebra, such as $\neg r(\sqrt{2})$ and $r((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})$.

$$\dfrac{\text{(LEM)}\ \dfrac{}{r(\sqrt{2}^{\sqrt{2}}) \vee \neg r(\sqrt{2}^{\sqrt{2}})} \qquad \nabla_1 \qquad \nabla_2}{\exists_x\exists_y(\neg r(x) \wedge \neg r(y) \wedge r(x^y))}(\vee_e)\ a, b$$

where ∇_1 is given by

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg r(\sqrt{2}) \qquad \dfrac{\neg r(\sqrt{2}) \qquad [r((\sqrt{2})^{\sqrt{2}})]^a}{\neg r(\sqrt{2}) \wedge r((\sqrt{2})^{\sqrt{2}})}(\wedge_i)}{\neg r(\sqrt{2}) \wedge \neg r(\sqrt{2}) \wedge r((\sqrt{2})^{\sqrt{2}})}(\wedge_i)}{\exists_y(\neg r(\sqrt{2}) \wedge \neg r(y) \wedge r((\sqrt{2})^y))}(\exists_i)}{\exists_x\exists_y(\neg r(x) \wedge \neg r(y) \wedge r(x^y))}(\exists_i)$$

and ∇_2 is given by

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\neg r(\sqrt{2}) \quad r((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})}{\neg r(\sqrt{2}) \wedge r((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})}(\wedge_i) \quad [\neg r(\sqrt{2}^{\sqrt{2}})]^b}{\neg r(\sqrt{2}^{\sqrt{2}}) \wedge \neg r(\sqrt{2}) \wedge r((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})}(\wedge_i)}{\exists_y(\neg r(\sqrt{2}^{\sqrt{2}}) \wedge \neg r(y) \wedge r((\sqrt{2}^{\sqrt{2}})^y))}(\exists_i)}{\exists_x\exists_y(\neg r(x) \wedge \neg r(y) \wedge r(x^y))}(\exists_i)$$

In the proof above, the witnesses depend on whether $\sqrt{2}^{\sqrt{2}}$ is rational or not. In the positive case, taking $x = y = \sqrt{2}$ allows us to conclude that x^y is rational, and in the negative case, this conclusion is achieved by taking $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$. So we proved the "existence" of an object without knowing explicitly the witnesses for x and y. This is acceptable as a proof in the classical logic, but not in the intuitionistic one.

Analogously to the intuitionistic case, the rules of the classical predicate logic are given by the rule schemes for the connectives ($\wedge$, $\vee$, $\neg$ and $\rightarrow$), the classical absurdity rule (PBC) (see Table 1.3) and the rules for the quantifiers (Table 2.1).

Example 13 While the sequents $\vdash \exists_x\varphi \rightarrow \neg\forall_x\neg\varphi$ and $\vdash \forall_x\varphi \rightarrow \neg\exists_x\neg\varphi$ have intuitionistic (indeed minimal) proofs as shown in Examples 11 and 12, the sequents $\vdash \neg\exists_x\neg\varphi \rightarrow \forall_x\varphi$ and $\vdash \neg\forall_x\neg\varphi \rightarrow \exists_x\varphi$ *have only* classical proofs. A proof for the former is given below.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\neg\exists x\neg\varphi]^u \quad \dfrac{[\neg\varphi[x/y]]^v}{\exists_x\neg\varphi}(\exists_i)}{\bot}(\neg_e)}{\varphi[x/y]}(\text{PBC})\ v}{\forall_x\varphi}(\forall_i)}{\neg\exists_x\neg\varphi \rightarrow \forall_x\varphi}(\rightarrow_i)\ u$$

Moreover, note that the above proof jointly with the one given in Example 11 shows that $\forall_x\varphi \dashv\vdash \neg\exists_x\neg\varphi$.

A proof of the sequent $\vdash \neg\forall_x\neg\varphi \rightarrow \exists_x\varphi$ is given below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\varphi[x/y]]^{v}}{\exists_x\varphi}\,(\exists_i) \qquad [\neg\exists_x\varphi]^{u}}{\bot}\,(\neg_e)}{\neg\varphi[x/y]}\,(\neg_i)\ v}{\forall_x\neg\varphi}\,(\forall_i) \qquad [\neg\forall_x\neg\varphi]^{w}}{\bot}\,(\neg_e)}{\exists_x\varphi}\,(\mathrm{PBC})\ u}{\neg\forall_x\neg\varphi \to \exists_x\varphi}\,(\to_i)\ w$$

Finally, this proof jointly with the one given in Example 12 shows that $\exists_x\varphi \dashv\vdash \neg\forall_x\neg\varphi$.

To verify that there are no possible intuitionistic derivations, notice that $\neg\exists_x\neg\varphi \to \forall_x\varphi$ and $\neg\forall_x\neg\varphi \to \exists_x\varphi$ together with the intuitionistic (indeed minimal) deduction rules allows derivation of non-intuitionistic theorems such as $\neg\neg\varphi \vdash \varphi$ (see next Exercise 28).

Exercise 28 Prove that there exist derivations for $\neg\neg\varphi \vdash \varphi$ using only the minimal natural deduction rules and each of the assumptions

a. $\neg\exists_x\neg\varphi \to \forall_x\varphi$ and
b. $\neg\forall_x\neg\varphi \to \exists_x\varphi$.

Hint: you can choose the variable x as any variable that does not occurs in φ. Thus, the application of rule $(\exists_e)$ over the existential formula $\exists_x\varphi$ has as witness assumption $[\varphi[x/x_0]]^{w}$ that has no occurrences of x_0.

In Exercise 24 we prove that there are intuitionistic derivations for $\neg\exists_x\varphi \dashv\vdash \forall_x\neg\varphi$. Also, in Example 12 we give an intuitionistic derivation for $\exists_x\neg\varphi \vdash \neg\forall_x\varphi$. Indeed, one can obtain minimal derivations for these three sequents.

Exercise 29 To complete $\neg\forall_x\varphi \dashv\vdash \exists_x\neg\varphi$ (see Example 12), prove that $\neg\forall_x\varphi \vdash \exists_x\neg\varphi$.

2.4 Semantics of the Predicate Logic

As done for the propositional logic in Chap. 1, here we present the standard semantics of first-order classical logic. The semantics of the predicate logic is not a direct extension of the one of propositional logic. Although this is not surprising, since the predicate logic has a richer language, there are some interesting points concerning the differences between propositional and predicate semantics that will be examined in this section. In fact, while a propositional formula has only finitely many interpretations, a predicate formula can have infinitely many ones.

We start with an example: let p be a unary predicate symbol, and consider the formula $\forall_x p(x)$. The variable x ranges over a domain, say the set of natural numbers $\mathbb{N}$. Is this formula true or false? Certainly, it depends on how the predicate symbol p is interpreted. If one interprets $p(x)$ as "x is a prime number", then it is false, but if $p(x)$ means that "x is a natural number" then it is true. Observe that the interpretation depends on the chosen domain, and hence the latter interpretation of p will be false over the domain of integers $\mathbb{Z}$.

This situation is similar in the propositional logic: according to the interpretation, some formulas can be either true or false. So what do we need to determine the truth value of a predicate formula? First of all, we need a domain of concrete individuals, i.e., a nonempty set $\mathtt{D}$ that represents known individuals (e.g., numbers, people, organisms, etc.). Function symbols (and constants) are associated to functions in the so called *structures*:

Definition 22 (*Structure*) A structure of a first-order language L over the set $S = (\mathbb{F}, \mathbb{P})$, also called an S-structure, is a pair $\langle \mathtt{D}, \mathtt{m} \rangle$, where $\mathtt{D}$ is a nonempty set and $\mathtt{m}$ is a map defined as follows:

1. if f is a function symbol of arity $n \geq 0$, then $\mathtt{m}(f)$ is a function from $\mathtt{D}^n$ to $\mathtt{D}$. A function from $\mathtt{D}^0$ to $\mathtt{D}$ is simply an element of $\mathtt{D}$.
2. if p is a predicate symbol of arity $n > 0$, then $\mathtt{m}(p)$ is a subset of $\mathtt{D}^n$.

Intuitively, the set $\mathtt{m}(p)$ contains the tuples of elements that satisfy the predicate p. As an example, consider the formula $q(a)$, where a is a constant, and the structure $\langle \{0, 1\}, \mathtt{m} \rangle$, where $\mathtt{m}(a) = 0$ and $\mathtt{m}(q) = \{0\}$. The formula $q(a)$ is true in this structure because the set $\mathtt{m}(q)$ contains the element 0, the image of the constant a by the function $\mathtt{m}$. But $q(a)$ would be false in other structures; for instance, it is false in the structure $\langle \{0, 1\}, \mathtt{m}' \rangle$, where $\mathtt{m}'(a) = 0$ and $\mathtt{m}'(q) = \emptyset$.

If a formula contains (free) variables, such as the formula $q(x)$, then a special mechanism is needed to interpret variables. Variables are associated to elements of the domain $\mathtt{D}$ through *assignments* that are functions from the set of variables $\mathbb{V}$ to the domain $\mathtt{D}$. So, if d is an assignment such that $d(x) = 0$ then $q(x)$ is true in the structure $\langle \{0, 1\}, \mathtt{m} \rangle$ above, and if $d'(x) = 1$ then $q(x)$ is false.

Definition 23 (*Interpretation of terms*) An *interpretation* I is a pair $\langle \langle \mathtt{D}, \mathtt{m} \rangle, d \rangle$ containing a structure and an assignment. Given an interpretation I and a term t, the interpretation of t by I, written t^I, is inductively defined as follows:

1. For each variable x, $x^I = d(x)$;
2. For each function symbol f with arity $n \geq 0$, $f(t_1, \ldots, t_n)^I = \mathtt{m}(f)(t_1^I, \ldots, t_n^I)$.

Thus, based on the interpretations of terms, the semantics of predicate formulas concerns the truth value of a formula that can be either T (true) or F (false). This notion is formalized in the following definition.

Definition 24 (*Interpretation of Formulas*) The truth value of a predicate formula φ according to a given interpretation of terms $I = \langle \langle \mathtt{D}, \mathtt{m} \rangle, d \rangle$, denoted as φ^I, is inductively defined as:

1. $\bot^I = F$ and $\top^I = T$;
2. $p(t_1, \ldots, t_n)^I = \begin{cases} T, & \text{if } (t_1^I, \ldots, t_n^I) \in \mathtt{m}(p), \\ F, & \text{if } (t_1^I, \ldots, t_n^I) \notin \mathtt{m}(p); \end{cases}$
3. $(\neg\psi)^I = \begin{cases} T, & \text{if } \psi^I = F, \\ F, & \text{if } \psi^I = T; \end{cases}$
4. $(\psi \wedge \gamma)^I = \begin{cases} T, & \text{if } \psi^I = T \text{ and } \gamma^I = T, \\ F, & \text{otherwise;} \end{cases}$
5. $(\psi \vee \gamma)^I = \begin{cases} T, & \text{if } \psi^I = T \text{ or } \gamma^I = T, \\ F, & \text{otherwise;} \end{cases}$
6. $(\psi \rightarrow \gamma)^I = \begin{cases} F, & \text{if } \psi^I = T \text{ and } \gamma^I = F, \\ T, & \text{otherwise;} \end{cases}$
7. $(\forall_x \psi)^I = \begin{cases} T, & \text{if } \psi^{I\frac{x}{a}} = T \text{ for every } a \in \mathtt{D}, \\ F, & \text{otherwise;} \end{cases}$
8. $(\exists_x \psi)^I = \begin{cases} T, & \text{if } \psi^{I\frac{x}{a}} = T \text{ for at least one } a \in \mathtt{D}, \\ F, & \text{otherwise.} \end{cases}$

where $I\frac{x}{a}$ denotes the interpretation I modifying its assignment d, in such a way that it maps x to a, and any other variable y to $d(y)$.

Definition 25 (*Models*) An interpretation I is said to be a *model* of φ if $\varphi^I = T$. We write $I \models \varphi$ to denote that I is a model of φ.

The notion of Model is extended to sets of formulas in a straightforward manner: If Γ is a set of predicate formulas then I is a model of Γ, denoted by $I \models \Gamma$, whenever I is a model of each formula in Γ.

Example 14 Let I be an interpretation with domain $\mathbb{N}$ and $\mathtt{m}(p) = \{(m, n) \in \mathbb{N} \times \mathbb{N} \mid m < n\}$. Then I is a model of $\forall_x \exists_y p(x, y)$, denoted as $I \models \forall_x \exists_y p(x, y)$, because for every natural x one can find another natural y bigger than x. With similar arguments, one can conclude that I is not a model of $\exists_x \forall_y p(x, y)$.

Definition 26 (*Satisfiability*) Let φ be a predicate formula. If φ has a model then it is said to be *satisfiable*; otherwise, it is *unsatisfiable*. This notion is also extended to sets of formulas: Γ is satisfiable if and only if there exist an interpretation I such that for all $\varphi \in \Gamma$, $I \models \varphi$.

Definition 27 (*Logical consequence and Validity*) Let $\Gamma = \{\phi_1, \ldots, \phi_n\}$ be a finite set of predicate formulas, and φ a predicate formula. We say that φ is a logical consequence of Γ, denoted as $\Gamma \models \varphi$, if every model of Γ is also a model of φ, i.e. $I \models \Gamma$ implies $I \models \varphi$, for every interpretation I. When Γ is empty then φ is said to be *valid*, which is denoted as $\models \varphi$.

Example 15 We claim that $\forall_x(p(x) \to q(x)) \models (\forall_x p(x)) \to (\forall_x q(x))$. In fact, let $I = \langle\langle \mathtt{D}, \mathtt{m}\rangle, d\rangle$ be a model of $\forall_x(p(x) \to q(x))$, i.e., $I \models \forall_x(p(x) \to q(x))$. If there exists an element in the domain of I that does not satisfy the predicate p then $\forall_x p(x)$ is false in I and hence, $(\forall_x(p(x)) \to (\forall_x q(x))$ would be true in I. Otherwise, $I \models \forall_x p(x)$, and hence $I\frac{x}{a} \models p(x)$, for all $a \in \mathtt{D}$. Since $I \models \forall_x(p(x) \to q(x))$, we conclude that $I\frac{x}{a} \models q(x)$, for all $a \in \mathtt{D}$. Therefore, $I \models \forall_x q(x)$.

The study of models can be justified by the fact that validity in a model is an invariant of provability in the sense that a sequent is provable exactly when all its interpretations are also models. This suggests a way to prove when a sequent is not provable: it is enough to find an interpretation that is not a model of the sequent. In the next section, we formalize this for the predicate logic.

2.5 Soundness and Completeness of the Predicate Logic

2.5.1 Soundness of the Predicate Logic

The soundness of predicate logic can be proved following the same idea used for the propositional logic. Therefore, we need to prove the following theorem:

Theorem 6 (Soundness of the predicate logic) *Let* Γ *be a set of predicate formulas, if* $\Gamma \vdash \varphi$ *then* $\Gamma \models \varphi$. *In other words, if* φ *is provable from* Γ *then* φ *is a logical consequence of* Γ.

Proof The proof is by induction on the derivation of $\Gamma \vdash \varphi$ similarly to the propositional case, and hence we focus just on the new rules: $(\forall_e)$, $(\forall_i)$, $(\exists_e)$, $(\exists_i)$.

If the last rule applied in the proof $\Gamma \vdash \varphi$ is $(\forall_e)$, then $\varphi = \psi[x/t]$ and the premise of the last rule is $\forall_x \psi$ as depicted in the following figure, where $\{\gamma_1, \ldots, \gamma_n\}$ is the subset of formulas in Γ used in the derivation.

$$\dfrac{\gamma_1 \quad \cdots \quad \gamma_n}{\dfrac{\forall_x \psi}{\psi[x/t]}\,(\forall_e)}$$

The subtree rooted by the formula $\forall_x \psi$ and with open leaves labeled by formulas in Γ, corresponds to a derivation for the sequent $\Gamma \vdash \forall_x \psi$ that by induction hypothesis implies $\Gamma \models \forall_x \psi$. Therefore, for all interpretations that make the formulas in Γ true, also $\forall_x \psi$ would be true: $I \models \Gamma$ implies $I \models \forall_x \psi$. The last implies that for all $a \in \mathtt{D}$, where $\mathtt{D}$ is the domain of I, $I\frac{x}{a} \models \psi$, and in particular, $I\frac{x}{t^I} \models \psi$. Consequently,

$I \models \psi[x/t]$. Therefore, one has that for any interpretation I, such that $I \models \Gamma$, $I \models \psi[x/t]$, which implies $\Gamma \models \psi[x/t]$.

If the last rule applied in the proof of $\Gamma \vdash \varphi$ is $(\forall_i)$, then $\varphi = \forall_x \psi$ and the premise of the last rule is $\psi[x/x_0]$ as depicted in the following figure:

$$\dfrac{\begin{array}{c}\gamma_1 \quad \cdots \quad \gamma_n \\ \hline \psi[x/x_0]\end{array}}{\forall_x \psi}\ (\forall_i)$$

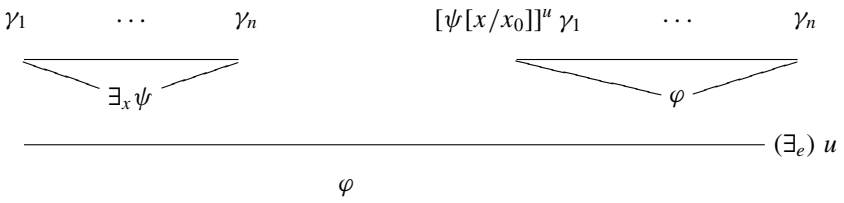
The subtree rooted by the formula $\psi[x/x_0]$ and with open leaves labeled by formulas in $\{\gamma_1, \ldots, \gamma_n\} \subset \Gamma$, corresponds to a derivation for the sequent $\Gamma \vdash \psi[x/x_0]$, in which no open assumption contains the variable x_0. This variable can be selected in such a manner that it does not appear free in any formula of Γ. By induction hypothesis, we have that $\Gamma \models \psi[x/x_0]$. This implies that all interpretations that make the formulas in Γ true, also make $\psi[x/x_0]$ true: $I \models \Gamma$ implies $I \models \psi[x/x_0]$. Since x_0 does not occurs in Γ, for all $a \in \mathtt{D}$, where $\mathtt{D}$ is the domain of I, $I\frac{x}{a} \models \Gamma$ and also $I\frac{x_0}{a} \models \psi[x/x_0]$ or, equivalently, $I\frac{x}{a} \models \psi$. Hence $\Gamma \models \forall_x \psi$.

If the last rule applied in the proof of $\Gamma \vdash \varphi$ is $(\exists_i)$, then $\varphi = \exists_x \psi$ and the premise of the last rule is $\psi[x/t]$ as depicted in the following figure, where again $\{\gamma_1, \ldots, \gamma_n\}$ is the subset of formulas of Γ used in the derivation:

$$\dfrac{\begin{array}{c}\gamma_1 \quad \cdots \quad \gamma_n \\ \hline \psi[x/t]\end{array}}{\exists_x \psi}\ (\exists_i)$$

The subtree rooted by the formula $\psi[x/t]$ and with open leaves labeled by formulas of Γ, corresponds to a derivation of the sequent $\Gamma \vdash \psi[x/t]$ that by induction hypothesis implies $\Gamma \models \psi[x/t]$. Therefore, any interpretation I that makes the formulas in Γ true, also makes $\psi[x/t]$ true. Thus, since $I \models \psi[x/t]$ implies $I\frac{x}{I^t} \models \psi$, one has that $I \models \exists_x \psi$. Therefore, $\Gamma \models \exists_x \psi$.

Finally, for a derivation of the sequent $\Gamma \vdash \varphi$ that finishes with an application of the rule $(\exists_e)$, one has as premises the formulas $\exists_x \psi$ and φ. The former labels a root of a subtree with open leaves labeled by assumptions in $\{\gamma_1, \ldots, \gamma_n\} \subset \Gamma$ that corresponds to a derivation for the sequent $\Gamma \vdash \exists_x \psi$; the later labels a subtree with open leaves in $\{\gamma_1, \ldots, \gamma_n\} \cup \{\psi[x/x_0]\}$ and corresponds to a derivation for the sequent $\Gamma, \psi[x/x_0] \vdash \varphi$, where x_0 is a variable that does not occur free in $\Gamma \cup \{\varphi\}$, as depicted in the figure below:



By induction hypothesis, one has $\Gamma \models \exists_x \psi$ and $\Gamma, \psi[x/x_0] \models \varphi$. The first means that for any interpretation I such that $I \models \Gamma$, $I \models \exists_x \psi$. Thus, there exists some $a \in \mathtt{D}$, the domain of I, such that $I\frac{x}{a} \models \psi$. Notice also that since x_0 does not occur in Γ, one has that $I\frac{x_0}{a} \models \Gamma$. From the second, since $I\frac{x_0}{a} \models \Gamma$, $\psi[x/x_0]$, one has that $I\frac{x_0}{a} \models \varphi$. But, since x_0 does not occurs in φ, one concludes that $I \models \varphi$. □

Exercise 30 Complete all other cases of the proof of the Theorem 6 of soundness of predicate logic.

2.5.2 Completeness of the Predicate Logic

The completeness proof for the predicate logic is not a direct extension of the completeness proof for the propositional logic. The completeness theorem was first proved by Kurt Gödel, and here we present the general idea of a proof due to Leon Albert Henkin (for nice complete presentations see references mentioned in the chapter on suggested readings).

The kernel of the proof is based on the fact that *every consistent set of formulas is satisfiable*, where consistency of the set Γ means that the absurd is not derivable from Γ:

Definition 28 A set Γ of predicate formulas is *consistent* if not $\Gamma \vdash \bot$.

Note that if we assume that **every consistent set is satisfiable** then the completeness can be easily obtained as follows:

Theorem 7 (Completeness) *Let Γ be a set of predicate formulas. If $\Gamma \models \varphi$ then $\Gamma \vdash \varphi$.*

Proof We prove that not $\Gamma \vdash \varphi$ implies not $\Gamma \models \varphi$. From not $\Gamma \vdash \varphi$ one has that $\Gamma \cup \{\neg\varphi\}$ is consistent because if $\Gamma \cup \{\neg\varphi\}$ were inconsistent then $\Gamma \cup \{\neg\varphi\} \vdash \bot$ by definition, and one could prove φ as follows:

$$\dfrac{\begin{array}{c}\Gamma, [\neg\varphi]^a \\ \vdots \\ \bot\end{array}}{\varphi}\ (\text{PBC})\ a$$

Therefore, $\Gamma \vdash \varphi$, which contradicts the supposition that not $\Gamma \vdash \varphi$. Now, since $\Gamma \cup \{\neg\varphi\}$ is consistent, by the assumption that consistent sets are satisfiable, we have that $\Gamma \cup \{\neg\varphi\}$ is satisfiable. Therefore, we conclude that not $\Gamma \models \varphi$. □

Our goal from now on is to prove that **every consistent set of formulas is satisfiable**. The idea is, given a consistent set of predicate formulas Γ, to build a model I for Γ, and since the sole available information is its consistency, this must be done by purely syntactical means that is using the language to build the desired model.

The key concepts in Henkin's proof are the notion of *witnesses* of existential formulas and extension of consistent sets of formulas to *maximally consistent* sets.

Definition 29 (*Witnesses and maximally consistency*) Let Γ be a set of formulas

Γ *contains witnesses* if and only if for every formula of the form $\exists_x \varphi$ in Γ, there exists a term t such that $\Gamma \vdash \exists_x \varphi \rightarrow \varphi[x/t]$.

Γ is *maximally consistent* if and only if for each formula φ, $\Gamma \vdash \varphi$ or $\Gamma \vdash \neg\varphi$.

Notice that from the definition, for any possible extension of a maximally consistent set Γ, say Γ' such that $\Gamma \subseteq \Gamma'$, $\Gamma' = \Gamma$. Maximally consistent sets are also said to be *closed for negation*.

The proof is done in two steps, and uses the fact that every subset of a satisfiable set is also satisfiable:

1. every consistent set can be extended to a maximally consistent set containing witnesses;
2. every maximally consistent set containing witnesses has a model.

If Γ does not contain witnesses, these formulas cannot be built in a straightforward manner, since one cannot choose any arbitrary term t to be witness of the existential formula without changing the semantics. Nevertheless, any consistent set can be extended to another consistent set containing witnesses. The simplest case is when the language is countable and the set Γ uses only a finite set of free variables that is $\mathtt{fv}(\Gamma)$ is finite. Since the set of existential formulas is also countable and there are infinite unused variable (those that do not appear free in Γ). Then these variables can be used as witnesses without any conflict. The other cases are more elaborated and are left as research exercises to the reader (Exercises 32 and 33): the case in which the language is countable, but Γ uses infinitely many free variables and the case in which the language is not countable.

In the sequel we will treat the simplest case in which the set of constant, function, and predicate symbols occurring in Γ is at most countable and there are only finitely many variables occurring in Γ. The next two lemmas complete the first part of the proof: a consistent set might be extended to a maximally consistent set with witnesses. This is done proving first how variables might be used to include witnesses and then how a consistent set with witnesses can be extended to a maximally consistent set.

Lemma 4 (Construction of witnesses) *Let Γ be a consistent set over a countable language such that $\mathtt{fv}(\Gamma)$ is finite. There exists an extension $\Gamma' \supseteq \Gamma$ over the same language, such that Γ' is consistent and contains witnesses.*

Proof Let $\exists_{x_1}\varphi_1, \exists_{x_2}\varphi_2, \ldots$ be an enumeration of all the existential formulas built over the language. Let $y_1, y_2, \ldots$ be an enumeration of the variables not occurring free in Γ, and consider the formulas below, for $i > 0$:

$$(\exists_{x_i}\varphi_i) \to \varphi_i[x_i/y_i]$$

Let Γ_0 be defined as Γ, and Γ_n, for $n > 0$ be defined as shown below:

$$\Gamma_n = \Gamma_{n-1} \cup \{(\exists_{x_n}\varphi_n) \to \varphi_n[x_n/y_n]\}$$

We will prove the consistence of Γ' defined as $\Gamma' = \bigcup_{n\in\mathbb{N}} \Gamma_n$ by induction on n. The base case is trivial since Γ is consistent by hypothesis. For $k > 0$, suppose Γ_{k-1} is consistent, but Γ_k is not, i.e.

$$\Gamma_k = \Gamma_{k-1} \cup \{(\exists_{x_k}\varphi_k) \to \varphi_k[x_k/y_k]\} \vdash \bot \tag{2.1}$$

Now consider the following derivation:

$$\dfrac{\text{(LEM) } (\exists_{x_k}\varphi_k) \vee \neg(\exists_{x_k}\varphi_k) \qquad \begin{array}{c}\Gamma_{k-1}\ [\exists_{x_k}\varphi_k]^a \\ \nabla_1 \\ \bot\end{array} \qquad \begin{array}{c}\Gamma_{k-1}\ [\neg\exists_{x_k}\varphi_k]^b \\ \nabla_2 \\ \bot\end{array}}{\bot}\ (\vee e)\, a\, b$$

where

$$\nabla_1\text{:}\quad \dfrac{[\exists_{x_k}\varphi_k]^a \qquad \dfrac{\Gamma_{k-1} \qquad \dfrac{[\varphi_k[x_k/y_k]]^u}{\exists_{x_k}\varphi_k \to \varphi_k[x_k/y_k]}\ (\to i)\emptyset}{\bot}\ (2.1)}{\bot}\ (\exists e)u$$

and

$$\nabla_2\text{:}\quad \dfrac{\Gamma_{k-1} \qquad \dfrac{\dfrac{\dfrac{[\neg\exists_{x_k}\varphi_k]^b}{\neg\varphi_k[x_k/y_k] \to \neg\exists_{x_k}\varphi_k}\ (\to i)\emptyset}{\exists_{x_k}\varphi_k \to \varphi_k[x_k/y_k]}\ (\text{CP})}{\bot}\ (2.1)$$

But this is a proof of $\Gamma_{k-1} \vdash \bot$ which contradicts the assumption that Γ_{k-1} is consistent. Therefore, Γ_k is consistent. □

In the previous proof, note that if $\Gamma_{i-1} \vdash \exists_{x_i} \varphi_i$ then it must be the case that $\Gamma_i \vdash \varphi_i[x_i/y_i]$ in order to preserve the consistency. Therefore, $\varphi_i[x_i/y_i]$ might be added to the set of formulas, but not its negation, as will be seen in the further construction of maximally consistent sets.

Now we prove that every maximally consistent set containing witnesses has a model.

Lemma 5 (Lindenbaum) *Each consistent set of formulas* Γ *over a countable language is contained in a maximally consistent set* Γ^* *over the same language.*

Proof Let $\delta_1, \delta_2, \ldots$ be an enumeration of the formulas built over the language. In order to build a consistent expansion of Γ we recursively define the family of indexed sets of formulas Γ_i as follows:

- $\Gamma_0 = \Gamma$
- $\Gamma_i = \begin{cases} \Gamma_{i-1} \cup \{\delta_i\}, & \text{if } \Gamma_{i-1} \cup \{\delta_i\} \text{ is consistent;} \\ \Gamma_{i-1}, & \text{otherwise.} \end{cases}$

Now let $\Gamma^* = \bigcup_{i \in \mathbb{N}} \Gamma_i$. We claim that Γ^* is maximally consistent. In fact, if Γ^* is not maximally consistent then there exists a formula $\gamma \notin \Gamma^*$ such that $\Gamma^* \cup \{\gamma\}$ is consistent. But by the above enumeration, there exists $k \geq 1$ such that $\gamma = \delta_k$, and since $\Gamma_{k-1} \cup \{\gamma\}$ should be consistent, $\delta_k \in \Gamma_{k+1}$. Hence $\delta_k = \gamma \in \Gamma^*$. □

From the previous Lemmas 4 and 5, one has that every consistent set of formulas built over a countable set of symbols and with finitely many free variables can be extended to a maximally consistent set which contains witnesses. In this manner we complete the first step of the prove.

Now, we will complete the second step of the proof that is that any maximally consistent set that contain witnesses is satisfiable. We start with two auxiliary definitional observations.

Lemma 6 *Let* Γ *be a maximally consistent set of formulas. Then for any formula* φ *either* $\varphi \in \Gamma$ *or* $\neg\varphi \in \Gamma$.

Lemma 7 *Let* Γ *be a maximally consistent set. For any formula* φ, $\Gamma \vdash \varphi$ *if, and only if* $\varphi \in \Gamma$.

Proof Suppose $\Gamma \vdash \varphi$. From Lemma 6, either $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$. If $\neg\varphi \in \Gamma$ then Γ would be inconsistent:

$$\dfrac{\begin{array}{cc} \Gamma & \Gamma \\ \nabla & \nabla \\ \varphi & \neg\varphi \end{array}}{\bot} (\neg_e)$$

Therefore, $\varphi \in \Gamma$. □

We now define a model that is called the *algebra* or *structure of terms* for the set Γ which is assumed to be maximally consistent and containing witnesses. The model, denoted as I_Γ, is built from Γ by taking as domain, the set $\mathtt{D}$ of all terms built over the countable language of Γ as given in the definition of terms Definition 13. The designation d for each variable is the same variable and the interpretation of each non-variable term is itself too: $t^{I_\Gamma} = t$. Notice that since our predicate language does not deal with equality symbol, different terms are interpreted as different elements of $\mathtt{D}$. The map $\mathtt{m}$ of I_Γ maps each n-ary function symbol in the language, f, in the function f^{I_Γ} such that for all terms $t_1, \ldots, t_n$, $(f(t_1, \ldots, t_n))^{I_\Gamma} = f^{I_\Gamma}(t_1^{I_\Gamma}, \ldots, t_n^{I_\Gamma}) = f(t_1, \ldots, t_n)$, and for each n-ary predicate symbol p, p^{I_Γ} is the relation defined as

$$(p(t_1, \ldots, t_n))^{I_\Gamma} = p^{I_\Gamma}(t_1^{I_\Gamma}, \ldots, t_n^{I_\Gamma}) \text{ if and only if } p(t_1, \ldots, t_n) \in \Gamma$$

With these definitions we have that for any atomic formula φ, $\varphi \in \Gamma$ if and only if $I_\Gamma \models \varphi$. In addition, according to the interpretation of quantifiers, for any atomic formula $\forall_{x_1} \ldots \forall_{x_n} \varphi \in \Gamma$ if and only if $I_\Gamma \models \forall_{x_1} \ldots \forall_{x_n} \varphi$ and $\exists_{x_1} \ldots \exists_{x_n} \varphi \in \Gamma$ if and only if $I_\Gamma \models \exists_{x_1} \ldots \exists_{x_n} \varphi$.

Using the assumptions that Γ has witnesses and is maximally consistent, formulas can be correctly interpreted in I_Γ as below.

1. $\bot^{I_\Gamma} = F$ and $\top^{I_\Gamma} = T$
2. $\varphi^{I_\Gamma} = T$, iff $\varphi \in \Gamma$, for any atomic formula φ
3. $(\neg\varphi)^{I_\Gamma} = T$, iff $\varphi^{I_\Gamma} = F$
4. $(\varphi \wedge \psi)^{I_\Gamma} = T$, iff $\varphi^{I_\Gamma} = T$ and $\psi^{I_\Gamma} = T$
5. $(\varphi \vee \psi)^{I_\Gamma} = T$, iff $\varphi^{I_\Gamma} = T$ or $\psi^{I_\Gamma} = T$
6. $(\varphi \rightarrow \psi)^{I_\Gamma} = T$, iff $\varphi^{I_\Gamma} = F$ or $\psi^{I_\Gamma} = T$
7. $(\exists_x \varphi)^{I_\Gamma} = T$, iff $(\varphi[x/t])^{I_\Gamma} = T$, for some term $t \in \mathtt{D}$
8. $(\forall_x \varphi)^{I_\Gamma} = T$, iff $(\varphi[x/t])^{I_\Gamma} = T$, for all $t \in \mathtt{D}$.

Indeed, this interpretation is well-defined only under the assumption that Γ has witnesses and is maximally consistent. For instance, the item 3 is well-defined since $\neg\varphi \in \Gamma$ if and only if not $\varphi \in \Gamma$. For the item 5, if $(\varphi \vee \psi) \in \Gamma$ and not $\varphi \in \Gamma$, by maximally consistency one has that $\neg\varphi \in \Gamma$; thus, from $(\varphi \vee \psi)$ and $\neg\varphi$, it is possible to derive ψ (by simple application of rules $(\vee_e)$ and $(\neg_e)$ and $(\bot_e)$). Similarly, if we assume $(\varphi \vee \psi) \in \Gamma$ and not $\psi \in \Gamma$, we can derive φ. For the item 6, suppose $(\varphi \rightarrow \psi) \in \Gamma$ and $\varphi \in \Gamma$, then one can derive ψ (by application of $(\rightarrow_e)$); otherwise, if $(\varphi \rightarrow \psi) \in \Gamma$ and not $\psi \in \Gamma$, by maximally consistency, $\neg\psi \in \Gamma$, from which one can infer $\neg\varphi$ (by application of contraposition). For the item 7, if we assume $\exists_x \varphi \in \Gamma$, by the existence of witnesses, there is a term t such that $\exists_x \varphi \rightarrow \varphi[x/t] \in \Gamma$, and from these two formulas we can derive $\varphi[x/t]$ (by a simple application of rule $(\rightarrow_e)$).

Exercise 31 Complete the analysis well-definedness for all the items in the interpretation of formulas I_Γ, for a set Γ that contains witnesses and is maximally complete.

Theorem 8 (Henkin) *Let* Γ *be a maximally consistent set containing witnesses. Then for all* φ*,*

$$I_\Gamma \models \varphi, \text{ if, and only if } \Gamma \vdash \varphi.$$

Proof The proof is done by induction on the structure of φ. If φ is an atomic formula then $\varphi \in \Gamma$ iff $(\varphi)^{I_\Gamma} = T$, by definition.
If $\varphi = \neg\varphi_1$ then

$$\begin{array}{ll} \neg\varphi_1 \in \Gamma & \Longleftrightarrow \text{ (because } \Gamma \text{ is maximally consistent)} \\ \varphi_1 \notin \Gamma & \Longleftrightarrow \text{ (by induction hypothesis)} \\ \text{not } I_\Gamma \models \varphi_1 & \Longleftrightarrow \text{ (by definition)} \\ I_\Gamma \models \neg\varphi_1. & \end{array}$$

If $\varphi = \varphi_1 \wedge \varphi_2$ then:

$$\begin{array}{ll} \varphi_1 \wedge \varphi_2 \in \Gamma & \Longleftrightarrow \text{ (by definition)} \\ \varphi_1 \in \Gamma \text{ and } \varphi_2 \in \Gamma & \Longleftrightarrow \text{ (by induction hypothesis for both } \varphi_1 \text{ and } \varphi_2) \\ I_\Gamma \models \varphi_1 \text{ and } I_\Gamma \models \varphi_2 & \Longleftrightarrow \text{ (by definition)} \\ I_\Gamma \models \varphi_1 \wedge \varphi_2. & \end{array}$$

If $\varphi = \varphi_1 \vee \varphi_2$ then:

$$\begin{array}{ll} \varphi_1 \vee \varphi_2 \in \Gamma & \Longleftrightarrow \text{ (by definition)} \\ \varphi_1 \in \Gamma \text{ or } \varphi_2 \in \Gamma & \Longleftrightarrow \text{ (by induction hypothesis for both } \varphi_1 \text{ and } \varphi_2) \\ I_\Gamma \models \varphi_1 \text{ or } I_\Gamma \models \varphi_2 & \Longleftrightarrow \text{ (by definition, no matter the condition holds for } \varphi_1 \text{ or } \varphi_2) \\ I_\Gamma \models \varphi_1 \vee \varphi_2. & \end{array}$$

If $\varphi = \varphi_1 \rightarrow \varphi_2$ then we split the proof into two parts. First, we show that $\varphi_1 \rightarrow \varphi_2 \in \Gamma$ implies $I_\Gamma \models \varphi_1 \rightarrow \varphi_2$. We have two subcases

1. $\varphi_1 \in \Gamma$: In this case, $\varphi_2 \in \Gamma$. In fact, if $\varphi_2 \notin \Gamma$ then $\neg\varphi_2 \in \Gamma$ by the maximality of Γ, and Γ becomes contradictorily inconsistent:

$$\dfrac{\dfrac{\varphi_1 \rightarrow \varphi_2 \qquad \varphi_1}{\varphi_2}\,(\rightarrow_e) \qquad \neg\varphi_2}{\bot}\,(\neg_e)$$

Thus, by induction hypothesis one has

$$\begin{array}{ll} \varphi_1 \in \Gamma \text{ and } \varphi_2 \in \Gamma & \Longleftrightarrow \text{ (by induction hypothesis for both } \varphi_1 \text{ and } \varphi_2) \\ I_\Gamma \models \varphi_1 \text{ and } I_\Gamma \models \varphi_2 & \Longrightarrow \text{ (by definition)} \\ I_\Gamma \models \varphi_1 \rightarrow \varphi_2. & \end{array}$$

2. $\varphi_1 \notin \Gamma$: In this case, $\neg\varphi_1 \in \Gamma$ by the maximality of Γ. Therefore,

$$\begin{array}{ll} \neg\varphi_1 \in \Gamma & \Longleftrightarrow \text{(by induction hypothesis)} \\ I_\Gamma \models \neg\varphi_1 & \Longleftrightarrow \text{(by definition)} \\ \text{not } I_\Gamma \models \varphi_1 & \Longrightarrow \text{(by definition)} \\ I_\Gamma \models \varphi_1 \to \varphi_2. & \end{array}$$

Now we prove that $I_\Gamma \models \varphi_1 \to \varphi_2$ implies $\varphi_1 \to \varphi_2 \in \Gamma$. By definition of the semantics of implication, there are two cases

1. $\varphi_1^{I_\Gamma} = F$: In this case, we have that $(\neg\varphi_1)^{I_\Gamma} = T$, and hence $\neg\varphi_1 \in \Gamma$, by induction hypothesis. We can now derive $\varphi_1 \to \varphi_2$ as follows, and conclude by Lemma 7:

$$\dfrac{\dfrac{\dfrac{\neg\varphi_1 \qquad [\varphi_1]^a}{\bot}\,(\neg_e)}{\varphi_2}\,(\bot_e)}{\varphi_1 \to \varphi_2}\,(\to_i)\ a$$

2. $\varphi_2^{I_\Gamma} = T$: By induction hypothesis $\varphi_2 \in \Gamma$, and we derive $\varphi_1 \to \varphi_2$ as follows, and conclude by Lemma 7

$$\dfrac{\varphi_2}{\varphi_1 \to \varphi_2}\,(\to_i)\ \emptyset$$

If $\varphi = \exists_x \varphi_1$ then

$$\begin{array}{ll} \exists_x \varphi_1 \in \Gamma & \Longleftrightarrow \text{(for some } t \in \mathtt{D}\text{, since } \Gamma \text{ contains witnesses)} \\ \varphi_1[x/t] \in \Gamma & \Longleftrightarrow \text{(by induction hypothesis)} \\ I_\Gamma \models \varphi_1[x/t] & \Longleftrightarrow \text{(by definition)} \\ I_\Gamma \models \exists_x \varphi_1. & \end{array}$$

If $\varphi = \forall_x \varphi_1$ then

$$\begin{array}{ll} \forall_x \varphi_1 \in \Gamma & \Longleftrightarrow \text{(otherwise } \Gamma \text{ becomes inconsistent as shown below)} \\ \varphi_1[x/t] \in \Gamma, \text{ for all } t \in \mathtt{D} & \Longleftrightarrow \text{(by induction hypothesis)} \\ I_\Gamma \models \varphi_1[x/t], \text{ for all } t \in \mathtt{D} & \Longleftrightarrow \text{(by definition)} \\ I_\Gamma \models \forall_x \varphi_1. & \end{array}$$

For the first equivalence, note that if $\forall_x \varphi_1 \in \Gamma$ then $\varphi_1[x/t] \in \Gamma$, for all term $t \in \mathtt{D}$, otherwise Γ becomes contradictorily inconsistent

$$\dfrac{\neg\varphi_1[x/t] \qquad \dfrac{\forall_x \varphi_1}{\varphi_1[x/t]}\,(\forall_e)}{\bot}\,(\bot_e)$$

□

Using as a model I_Γ, it is possible to conclude, in this case, that consistent sets are satisfiable.

Corollary 2 (Consistency implies satisfiability) *If* Γ *is a consistent set of formulas over a countable language with a finite set of free variables then* Γ *is satisfiable.*

Proof Initially, Γ is consistently enlarged obtaining the set Γ' including witnesses according to the construction in Lemma 4; afterwards, Γ' is closed maximally obtaining the set $(\Gamma')^*$according to the construction in Lindenbaum's Lemma 5. This set contains witnesses and is maximally consistent; then, by Henkin's Theorem 8, I_Γ is a model of $(\Gamma')^*$, hence a model of Γ too. □

Exercise 32 (*) Research in the suggested related references how a consistent set built over a countable set of symbols, but that uses infinite free variables can be extended to a maximal consistent set with witnesses. The problem is that in this case there are no new variables that can be used as witnesses. Thus, one needs to extend the language with new constant symbols that will act as witnesses, but each time a new constant symbol is added to the language the set of existential formulas change.

Exercise 33 (*) Research the general case in which the language is not restricted, that is the case in which Γ is built over a non-countable set of symbols.

2.5.3 Compactness Theorem and Löwenheim-Skolem Theorem

The connections between $\models$ and $\vdash$ as well as between consistence and satisfiability provided in this section, give rise to other additional important consequences that relate semantic and syntactic elements of the predicate logic. Here we present two important theorems that are related with the scope and limits of the expressiveness of predicate logic.

Theorem 9 (Compactness) *Given a set* Γ *of predicate formulas and a formula* φ*, the following holds:*

i. $\Gamma \models \varphi$ *if and only if there is a finite set* $\Gamma_0 \subseteq \Gamma$ *such that* $\Gamma_0 \models \varphi$
ii. Γ *is satisfiable if and only if for all finite set* $\Gamma_0 \subseteq \Gamma$*,* Γ_0 *is satisfiable.*

Proof i. For necessity, if $\Gamma \models \varphi$, by completeness there exists a derivation ∇ for $\Gamma \vdash \varphi$. The derivation ∇ uses only a finite subset of assumptions, say $\Gamma_0 \subseteq \Gamma$. Thus, $\Gamma_0 \vdash \varphi$ and, by correctness, one concludes that $\Gamma_0 \models \varphi$. For sufficiency, suppose that $\Gamma_0 \models \varphi$, for a finite set $\Gamma_0 \subseteq \Gamma$. By completeness there exists a derivation ∇ for $\Gamma_0 \vdash \varphi$. But ∇ is also a derivation for $\Gamma \vdash \varphi$; hence, by correctness one concludes that $\Gamma \models \varphi$.

ii. Necessity is proved by contraposition: if Γ_0 were unsatisfiable for some finite set $\Gamma_0 \subseteq \Gamma$, then Γ_0 would be inconsistent, since consistency implies satisfiability (Corollary 2); thus, $\Gamma_0 \vdash \bot$, which implies also that $\Gamma \vdash \bot$ and by correctness that $\Gamma \models \bot$. Hence, Γ would be unsatisfiable. Sufficiency is proved also by contraposition: if we assume that Γ is unsatisfiable, then since there exists no model for Γ, $\Gamma \models \bot$ holds. By completeness also, $\Gamma \vdash \bot$ and hence, there exists a finite set $\Gamma_0 \subseteq \Gamma$, such that $\Gamma_0 \vdash \bot$, which by correctness implies that $\Gamma_0 \models \bot$. Thus, we conclude that Γ_0 is unsatisfiable.

□

The compactness theorem has several applications that are useful for restricting the analysis of consistency and satisfiability of arbitrary sets of predicate formulas to only finite subsets. This also has important implications in the possible cardinality of models of sets of predicate formulas such as the one given in the following theorem.

Theorem 10 (Löwenheim–Skolem) *Let* Γ *be a set of formulas such that for any natural* $n \in \mathbb{N}$*, there exists a model of* Γ *with a domain of cardinality at least* n*. Then* Γ *has also infinite models.*

Proof Consider an additional binary predicate symbol E and the formulas φ_n for $n > 0$, defined as

$$\forall_x \; E(x, x) \wedge \exists_{x_1, \ldots, x_n} \bigwedge_{i \neq j; i, j = 1}^{n} \neg E(x_i, x_j)$$

For instance, the formulas φ_1 and φ_3 are given respectively as $\forall_x E(x, x)$ and $\forall_x \; E(x, x) \wedge \exists_{x_1} \exists_{x_2} \exists_{x_3} \; (\neg E(x_1, x_2) \wedge \neg E(x_1, x_3) \wedge \neg E(x_2, x_3))$.

Notice that φ_n has models of cardinality at least n. It is enough to interpret E just as a the reflexive relation among the elements of the domain of the interpretation. Thus, pairs of different elements of the domain do not belong to the interpretation of E.

Let Φ be the set of formulas $\{\varphi_n \mid n \in \mathbb{N}\}$. We will prove that all finite subsets of the set of formulas $\Gamma \cup \Phi$ are satisfiable and then by the compactness theorem conclude that $\Gamma \cup \Phi$ is satisfiable too. An interpretation $I \models \Gamma \cup \Phi$ should have an infinite model, since also $I \models \Phi$ and all formulas in Φ are true in I only if there are infinitely many elements in the domain of I.

To prove that any finite set $\Gamma_0 \subset \Gamma \cup \Phi$ is satisfiable, let k be the maximum k such that $\varphi_k \in \Gamma_0$. Since Γ has models of arbitrary finite cardinality, let I' be a model of Γ with at least k elements in its domain $\mathtt{D}$. I' can be extended in such a manner that the binary predicate symbol E is interpreted just as the reflexive relation over $\mathtt{D}$. Let I be the extended interpretation. It is clear that $I \models \Gamma$ since E is a new symbol and also $I \models \Gamma_0 \cap \Phi$ since the domain has at least k different elements. Also, since $I \models \Gamma$, we have that $I \models \Gamma \cap \Gamma_0$. Hence, $I \models \Gamma_0$ and so we conclude that Γ_0 is satisfiable. □

Exercise 34 Prove that there is no predicate formula φ that holds exclusively for all finite interpretations.

Exercise 35 Let E be a binary predicate symbol, e a constant and $\cdot$ and -1 be binary and unary function symbols, respectively. The theory of groups is given by the models of the set of formulas Γ_G

$$\begin{gathered}
\forall_x\ E(x,x) \\
\forall_{x,y}\ (E(x,y) \to E(y,x)) \\
\forall_{x,y,z}\ (E(x,y) \wedge E(y,z) \to E(x,z)) \\
\forall_x\ E(x \cdot e, x) \\
\forall_x\ E(x \cdot x^{-1}, e) \\
\forall_{x,y,z}\ E((x \cdot y) \cdot z, x \cdot (y \cdot z))
\end{gathered}$$

Notice that according to the three first axioms the symbol E should be interpreted as an equivalence relation such as the equality. Indeed, the three other axioms are those related with group theory itself: the fourth one states the existence of an identity element, the fifth one the inverse function and the sixth one the associativity of the binary operation.

Prove the existence of infinite models by proving that for any $n \in \mathbb{N}$, the structure of arithmetic modulo n is a group of cardinality n. The elements of this structure are all integers modulo n (i.e., the set $\{0, 1, \ldots, n-1\}$), with addition and identity element 0.

Exercise 36 A graph is a structure of the form $G = \langle V, E \rangle$, where V is a finite set of vertices and $E \subset V \times V$ a set of edges between the vertices. The problem of reachability in graphs is the question whether there exists a finite path of *consecutive* edges, say $(u, u_1), (u_1, u_2), \ldots, (u_{n-1}, v)$, between two given nodes $u, v \in V$.

Prove that there is no predicate formula that expresses reachability in graphs. Hint: the key observation to conclude is that the problem of reachability between two nodes might be answered positively whenever there exists a path of arbitrary length.

2.6 Undecidability of the Predicate Logic

The gain of expressiveness obtained in predicate logic w.r.t. to the propositional logic comes at a price. Initially, remember that for a given propositional formula φ, one can always answer whether φ is valid or not by analyzing its truth table. This means that there is an algorithm that receives an arbitrary propositional formula as input and **always** answers after a finite amount of time `yes`, if the given formula is valid; or `no`, otherwise. The algorithm works as follows: build the truth table for φ and check whether it is true for all interpretations. Note that this algorithm is not efficient because the (finite) number of possible interpretations grows exponentially w.r.t. the number of propositional variables occurring in φ.

In general, a computational question with a `yes` or `no` answer depending on the parameters is known as a *decision problem*. A decision problem is said to be *decidable* whenever there exists an algorithm that correctly answers `yes` or `no` for each instance

of the problem, and when such algorithm does not exist the decision problem is said to be *undecidable*. Therefore, we conclude that that *validity* is decidable in propositional logic.

The natural question that arises at this point is whether validity is decidable or not in predicate logic. Note that the truth table approach is no longer possible because the number of different interpretations for a given predicate formula φ is not finite. In fact, as stated in the previous paragraph the gain of expressiveness of the predicate logic comes at a price: validity is undecidable in predicate logic. This fact is usually known as the *undecidability of predicate logic*, and has several important consequences. In fact, it is straightforward from the completeness of predicate logic that provability is also undecidable, i.e., there is no algorithm that receives a predicate formula φ as input and returns `yes` if $\vdash \varphi$, or `no` if not $\vdash \varphi$.

The standard technique for proving the undecidability of the predicate logic consists in reducing a known undecidable problem to the validity of the predicate logic in such a way that decidability of validity of the predicate logic entails the decidability of the other problem leading to a contradiction. In what follows, we consider the word problem for a specific monoid introduced by G.S. Tseitin, and that is well-known to be undecidable.

A semigroup is an algebraic structure with a binary associative operator $\cdot$ over a given set A. When in addition the structure has an identity element id which is called a monoid. By associativity, one understands that for all x, y, z in A, $x \cdot (y \cdot z) = (x \cdot y) \cdot z$, and for all $x \in A$ the identity satisfies the properties $id \cdot x = x$ and $x \cdot id = x$. In general, the word problem in a given semigroup with a given set of equations E (between pairs of elements of A), is the problem of answering whether two words are equal *applying* these equations.

By an application of an equation, say $u = v$ in E, one can understand an equational transformation of the form below, where $x\ y$ are any elements of A.

$$x \cdot (u \cdot y) = x \cdot (v \cdot y)$$

Hence, the word problem consists in answering for any pair of elements $x, y \in A$ if there exists a finite chain, possibly of length zero, of applications of equations that transform x in y:

$$x \equiv x_0 \overset{u_1=v_1}{=} x_1 \overset{u_2=v_2}{=} x_2 \overset{u_3=v_3}{=} \ldots \overset{u_n=v_n}{=} x_n \equiv y \tag{2.2}$$

In the chain above, the notation $\equiv$ is used for syntactic equality and $\overset{u_i=v_i}{=}$ for highlighting that the equation applied in the application step is $u_i = v_i$.

Tseitin's monoid is given by the set Σ^* of words freely generated by the quinary alphabet $\Sigma = \{a, b, c, d, e\}$. In this structure, the binary associative operator is the concatenation of words and the empty word plays the role of the identity. The set of equations is given below. For simplicity, we will omit parentheses and the concatenation operator.

$$\begin{aligned} ac &= ca \\ ad &= da \\ bc &= cb \\ bd &= db \\ ce &= eca \\ de &= edb \\ cdca &= cdcae \end{aligned} \tag{2.3}$$

As previously mentioned, Tseitin introduced this specific monoid with the congruence generated by this set of equations and proved that the word problem in this structure is undecidable.

In order to reduce the above problem to the validity of the predicate logic, we choose a logical language with a constant symbol $\Box$, five unary function symbols f_a, f_b, f_c, f_d, and f_e, and a binary predicate P. The constant $\Box$ will be interpreted as the empty word, and each function symbol, say $f_\star$ for $\star \in \Sigma$, as the concatenation of the symbol $\star$ to the left of the term given as argument of $f_\star$. For example, the word $baaecde$ will be encoded as $f_b(f_a(f_a(f_e(f_c(f_d(f_e(\Box)))))))$, which for brevity will be written simply as $f_{baaecde}(\Box)$. The binary predicate P will play the role of equality, i.e., $P(x, y)$ is interpreted as x is equal to y (modulo the congruence induced by the set of equations above, which would be assumed as axioms).

Our goal is, given an instance of the word problem $x, y \in \Sigma^*$ specified above, to build a formula $\varphi_{x,y}$ such that x equals y in this structure if and only if $\models \varphi_{x,y}$. The formula $\varphi_{x,y}$ is of the form

$$\varphi' \to P(f_x(\Box), f_y(\Box)) \tag{2.4}$$

where φ' is the following formula:

$$\begin{aligned} &\forall_x(P(x,x)) \wedge \\ &\forall_x \forall_y(P(x,y) \to P(y,x)) \wedge \\ &\forall_x \forall_y, \forall_z(P(x,y) \wedge P(y,z) \to P(x,z)) \wedge \\ &\forall_x \forall_y(P(x,y) \to P(f_{ac}(x), f_{ca}(y))) \wedge \\ &\forall_x \forall_y(P(x,y) \to P(f_{ad}(x), f_{da}(y))) \wedge \\ &\forall_x \forall_y(P(x,y) \to P(f_{bc}(x), f_{cb}(y))) \wedge \\ &\forall_x \forall_y(P(x,y) \to P(f_{bd}(x), f_{db}(y))) \wedge \\ &\forall_x \forall_y(P(x,y) \to P(f_{ce}(x), f_{eca}(y))) \wedge \\ &\forall_x \forall_y(P(x,y) \to P(f_{de}(x), f_{edb}(y))) \wedge \\ &\forall_x \forall_y(P(x,y) \to P(f_{cdca}(x), f_{cdcae}(y))) \wedge \\ &\forall_x \forall_y(P(x,y) \to P(f_a(x), f_a(y))) \wedge \\ &\forall_x \forall_y(P(x,y) \to P(f_b(x), f_b(y))) \wedge \end{aligned} \tag{2.5}$$

$$\forall_x \forall_y (P(x, y) \rightarrow P(f_c(x), f_c(y))) \wedge$$
$$\forall_x \forall_y (P(x, y) \rightarrow P(f_d(x), f_d(y))) \wedge$$
$$\forall_x \forall_y (P(x, y) \rightarrow P(f_e(x), f_e(y)))$$

Suppose $\models \varphi_{x,y}$. Our goal is to find a model for $\varphi_{x,y}$ which tells us if there is a solution to the instance $x, y \in \Sigma^*$. Consider the interpretation I with domain Σ^* and such that

- the constant $\square$ is interpreted as the empty word;
- each unary function symbol $f_\star$, for $\star \in \Sigma$, is interpreted as the function $f_\star^I : \Sigma^* \rightarrow \Sigma^*$ that appends the symbol $\star$ to the word $x \in \Sigma^*$ given as argument, i.e., $f_\star^I(x) = \star x$;
- and the binary predicate P is interpreted as follows: $P(x, y)^I$ if and only if there exists a chain, possibly of length zero, of applications of the Eqs. (2.3) that transform x into the word y.

We claim that $I \models \varphi'$. Let us consider each case

- $I \models \forall x (P(x, x))$: take the empty chain.
- $I \models \forall_x \forall_y (P(x, y) \rightarrow P(y, x))$: for any x, y such that $I \models P(x, y)$, take the chain given for $P(x, y)$ in reverse order.
- $I \models \forall_x \forall_y \forall_z (P(x, y) \wedge P(y, z) \rightarrow P(x, z))$: for any x, y, z such that $I \models P(x, y)$ and $I \models P(y, z)$, append the chains given for $P(x, y)$ and $P(y, z)$.
- $I \models \forall_x \forall_y (P(x, y) \rightarrow P(f_{ac}(x), f_{ca}(y)))$: for any x, y such that $I \models P(x, y)$, take the chain given for $P(x, y)$ and use this for the chain of equations for $acx = acy$; then add an application of the equation $ac = ca$ to obtain cay. A similar justification is given for all other cases related with Eqs. (2.3), but the last.
- $I \models \forall_x \forall_y (P(x, y) \rightarrow P(f_\star(x), f_\star(y)))$ where $\star \in \Sigma$: for any x, y such that $I \models P(x, y)$, take the chain given for $P(x, y)$ and use it for the chain for the equation $\star x = \star y$.

Since $I \models \varphi_{x,y}$ and $I \models \varphi'$, we conclude that $I \models P(f_x(\square), f_y(\square))$. Therefore, the instance x, y of the word problem has a solution.

Conversely, suppose the instance x, y of the word problem has a solution in Tseitin's monoid; i.e., there is a chain of applications of the Eqs. (2.3) from x resulting in the word y as given in the chain (2.2). We will suppose that this chain is of length n.

We need to show that $\varphi_{x,y}$ is valid; i.e., that $\models \varphi_{x,y}$. Let us consider an arbitrary interpretation I' over a domain D with an element $\square^{I'}$, five unary functions $f_a^{I'}$, $f_b^{I'}$, $f_c^{I'}$, $f_d^{I'}$, $f_e^{I'}$ and a binary relation $P^{I'}$. Since $\varphi_{x,y}$ is equal to $\varphi' \rightarrow P(f_u(\square), f_v(\square))$, we have to show that if $I' \models \varphi'$ then $I' \models P(f_u(\square), f_v(\square))$.

We proceed by induction in n, the length of the chain of applications of Eqs. (2.3) for transforming x in y.

IB: case $n = 0$, we have that $x \equiv y$ and if $I' \models \varphi'$, $I' \models \forall_x P(x, x)$ which also implies that $I' \models P(x, x)$.

IS: case $n > 0$, the chain of applications of equations to transform x in y is of the form

$$x \equiv x_0 \overset{u_1=v_1}{=} x_1 \overset{u_2=v_2}{=} x_2 \overset{u_3=v_3}{=} \ldots x_{n-1} \overset{u_n=v_n}{=} x_n \equiv y$$

By induction hypothesis we have that $I' \models P(x, x_{n-1})$. If we prove that $I' \models P(x_{n-1}, y)$, we can conclude that $I' \models P(x, y)$, since $I' \models \forall_x \forall_y \forall_z P(x, y) \wedge P(y, z) \rightarrow P(x, z)$ because we are assuming that $I' \models \varphi'$.

Thus, the proof resumes to prove that equalities obtained by one step of application of equations in (2.3) hold in I': in particular if we suppose that $u_n = v_n$ is the equation $u = v$ in (2.3), $x_{n-1} \equiv wuz$ and $y = wvz$, we need to prove that $I' \models P(f_{wuz}(\Box), f_{wvz}(\Box))$, which is done by the following three steps:

1. First, one has that $I' \models P(f_z(\Box), f_z(\Box))$, since $I' \models \forall_x P(x, x)$.
2. Second, since $u = v$ in (2.3), $I' \models \forall_x \forall_y (P(x, y) \rightarrow P(f_u(x), f_v(y)))$. Thus, by the previous item one has that $I' \models P(f_{uz}(\Box), f_{vz}(\Box))$;
3. Third, $I' \models P(f_{wuz}(\Box), f_{wvz}(\Box))$ is obtained from the last item, inductively on the length of w, since $I' \models \forall_x \forall_y (P(x, y) \rightarrow P(f_\star(x), f_\star(y)))$, for all $\star \in \Sigma$.

To conclude the undecidability of validity of the predicate logic, if we suppose the contrary, we will be able to answer for any $x, y \in \Sigma^*$ if $\models P(f_x(\Box), f_y(\Box))$ answering consequently if x equals y in Tseitin's monoid, which is impossible since the word problem in this structure is undecidable.

Theorem 11 (Undecidability of the Predicate Logic) *Validity in the predicate logic that is answering whether for a given formula φ, $\models \varphi$ is undecidable.*

Notice that by Gödel completeness theorem undecidability of validity immediately implies undecidability of derivability in the predicate logic. Indeed, in the above reasoning one can use the completeness theorem to alternate between validity and derivability.

Exercise 37 Accordingly to the three steps above to prove $I' \models P(f_{wuz}(\Box), f_{wvz}(\Box))$, build a derivation for the sequent $\vdash P(f_{wuz}(\Box), f_{wvz}(\Box))$. Concretely, prove that

a. $\varphi' \vdash P(f_z(\Box), f_z(\Box))$, for $z \in \Sigma^*$;
b. $\varphi', P(f_z(\Box), f_z(\Box)) \vdash P(f_{uz}(\Box), f_{vz}(\Box))$, for $u = v$ in the set of equations (2.3);
c. $\varphi', P(f_{uz}(\Box), f_{vz}(\Box)) \vdash P(f_{wuz}(\Box), f_{wvz}(\Box))$, for $w \in \Sigma^*$;
d. $\varphi' \vdash P(f_{wuz}(\Box), f_{wvz}(\Box))$.

Chapter 3
Deductions in the Style of Gentzen's Sequent Calculus

In this chapter, we present a style of deduction known as Gentzen's sequent calculus that is different from the one of natural deduction (both invented by Gerhard Gentzen) and has relevant computational interest and applications. The goal of this section is to present the alternative for deduction *à la* Gentzen sequent calculus, proving its equivalence with Gentzen's natural deduction. This sequent style is the one used by the proof assistant PVS that will be used in the next chapter.

3.1 Motivation

Both deduction technologies, natural deduction and Gentzen's sequent calculus, were invented by the German mathematician Gerhard Gentzen in the 1930s, although it is known that the Polish logician Stanisław Jaśkowski was the first to present a system of natural deduction. In sequent calculi *à la* Gentzen (for short, we will use "calculus *à la* Gentzen" or "sequent calculus"), deductions are trees as in natural deduction, but instead formulas, nodes are labeled by *sequents* of the form:

$$\Gamma \Rightarrow \Delta$$

The sequent expresses that Δ is deducible from Γ, where Γ and Δ are sequences of formulas, or more precisely as we will see multisets indeed. The multiset Γ is called the *antecedent*, while Δ is the *succedent* of the sequent, or respectively, the premises and conclusions of the sequent.

From this point of view, Gentzen's sequent calculus can be interpreted as a meta calculus for systems of natural deduction. As a very simple example consider the sequent

$$\varphi \Rightarrow \varphi$$

According to the above interpretation, this means that φ can be deduced from φ. Indeed, in natural deduction one has a derivation for $\varphi \vdash \varphi$, which consists of a tree

M. Ayala-Rincón and F.L.C. de Moura, *Applied Logic for Computer Scientists*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-51653-0_3

of the form

$$[\varphi]^u$$

This derivation means that assuming φ, one concludes φ. In the sequent calculus the simplest rule is the axiom (Ax), that is a sequent with a formula, say φ, that occurs both in the antecedent and in the succedent:

$$\Gamma, \varphi \Rightarrow \varphi, \Delta \quad \text{(Ax)}$$

As a second simple example, consider the sequent

$$\varphi, \varphi \to \psi \Rightarrow \psi$$

This sequent means that ψ is deducible from φ and $\varphi \to \psi$. And in natural deduction one has the corresponding derivation depicted as the tree:

$$\frac{[\varphi]^u \qquad [\varphi \to \psi]^v}{\psi} \ (\to_e)$$

Notice that in the informal interpretation of the sequent $\varphi, \varphi \to \psi \Rightarrow \psi$, it is expressed that the formula ψ in the succedent is derivable from the formulas in the antecedent. Correspondingly, in the natural derivation tree this is expressed by the two undischarged assumptions $[\varphi]^u$ and $[\varphi \to \psi]^v$ and the conclusion ψ.

As we will formally see, the corresponding proof-tree *à la* Gentzen sequent calculus is given by the following tree in which the rule $(\mathrm{L}_{\to})$, read as "*left* implication", is applied:

$$\frac{\varphi \Rightarrow \varphi \ \text{(Ax)} \qquad \psi \Rightarrow \psi \ \text{(Ax)}}{\varphi, \varphi \to \psi \Rightarrow \psi} \ (\mathrm{L}_{\to})$$

The intuition with rule $(\mathrm{L}_{\to})$ in this deduction is that whenever both φ is deducible from φ and ψ from ψ, ψ is deducible from φ and $\varphi \to \psi$.

From the computational point of view, proofs in a sequent calculus are trees that use more memory in their node labels than proofs in natural deduction. But one has the advantage that in each step of the deductive process all assumptions and conclusions are available directly in the current sequent under consideration, which makes unnecessary searching from assumptions (to be discharged or copied) in previous leaves of the proof-tree.

3.2 A Gentzen's Sequent Calculus for the Predicate Logic

As previously mentioned, sequents are expressions of the form $\Gamma \Rightarrow \Delta$, where Γ and Δ are finite *multisets* of formulas. A multiset is a set in which elements can appear repeatedly. Thus, formulas can appear repeatedly in Γ and Δ. The inference rules

Table 3.1 Axioms and structural rules of Gentzen's SC for predicate logic

Axioms:	
$\bot, \Gamma \Rightarrow \Delta \ \ (\mathrm{L}_\bot)$	$\Gamma, \varphi \Rightarrow \varphi, \Delta \ \ (\mathrm{Ax})$
left rules	right rules
Structural rules:	
$\dfrac{\Gamma \Rightarrow \Delta}{\varphi, \Gamma \Rightarrow \Delta}$ (LW*eakening*)	$\dfrac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \varphi}$ (RW*eakening*)
$\dfrac{\varphi, \varphi, \Gamma \Rightarrow \Delta}{\varphi, \Gamma \Rightarrow \Delta}$ (LC*ontraction*)	$\dfrac{\Gamma \Rightarrow \Delta, \varphi, \varphi}{\Gamma \Rightarrow \Delta, \varphi}$ (RC*ontraction*)

of the Gentzen sequent calculus for predicate logic are given in the Tables 3.1 and 3.2. The sequent deduction rules are divided into left ("L") and right ("R"), axioms, structural rules, and logical rules.

In these rules, Γ and Δ are called the *context* of the rule, the formula in the conclusion of the rule, not in the context, is called the *principal* formula, and the formulas in the premises of the rules, from which the principal formula derives, are called the *active* formulas. In rule (Ax) both occurrences of φ are principal and in $(\mathrm{L}_\bot)$ $\bot$ is principal.

An important observation is that in sequent calculus, the syntax does not include negation ($\neg$). Thus, there are no logical rules for negation in Gentzen's sequent calculus. Negation of a formula φ, that is $\neg\varphi$, would be used here as a shortcut for the formula $\varphi \to \bot$.

The weakening structural rules, for short denoted as (RW) and (LW), mean that whenever Δ holds from Γ, Δ holds from Γ *and* any other formula φ ((LW)) and, from Γ also Δ *or* any other formula φ hold ((RW)). In natural deduction, the intuitive interpretation of weakening rules is that if one has a derivation for $\Gamma \vdash \delta$, also a derivation for $\Gamma, \varphi \vdash \delta$ would be possible ((LW)); on the other side, from $\Gamma \vdash \delta$ one can infer a derivation for $\Gamma \vdash \delta \vee \varphi$ ((RW)). As we will see, some technicalities would be necessary to establish a formal correspondence since in sequent calculus we are working with sequents that are object different to formulas. Indeed, if Δ consists of more than one formula it makes no sense to search for a natural derivation with conclusion Δ.

The contraction structural rules, for short denoted as (RC) and (LC), mean that whenever Δ holds from the set φ, φ, Γ, then Δ still holds if one copy of the duplicated formula φ is deleted from it (case (LC)). On the right side, the analysis of the sequent

Table 3.2 Logical rules of Gentzen's sequent calculus for predicate logic

left rules	right rules
Logical rules:	
$\dfrac{\varphi_{i\in\{1,2\}}, \Gamma \Rightarrow \Delta}{\varphi_1 \wedge \varphi_2, \Gamma \Rightarrow \Delta}\ (\mathrm{L}_\wedge)$	$\dfrac{\Gamma \Rightarrow \Delta, \varphi \quad \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \wedge \psi}\ (\mathrm{R}_\wedge)$
$\dfrac{\varphi, \Gamma \Rightarrow \Delta \quad \psi, \Gamma \Rightarrow \Delta}{\varphi \vee \psi, \Gamma \Rightarrow \Delta}\ (\mathrm{L}_\vee)$	$\dfrac{\Gamma \Rightarrow \Delta, \varphi_{i\in\{1,2\}}}{\Gamma \Rightarrow \Delta, \varphi_1 \vee \varphi_2}\ (\mathrm{R}_\vee)$
$\dfrac{\Gamma \Rightarrow \Delta, \varphi \quad \psi, \Gamma \Rightarrow \Delta}{\varphi \to \psi, \Gamma \Rightarrow \Delta}\ (\mathrm{L}_\to)$	$\dfrac{\varphi, \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \to \psi}\ (\mathrm{R}_\to)$
$\dfrac{\varphi[x/t], \Gamma \Rightarrow \Delta}{\forall_x \varphi, \Gamma \Rightarrow \Delta}\ (\mathrm{L}_\forall)$	$\dfrac{\Gamma \Rightarrow \Delta, \varphi[x/y]}{\Gamma \Rightarrow \Delta, \forall_x \varphi}\ (\mathrm{R}_\forall), \quad y \notin \mathtt{fv}(\Gamma, \Delta)$
$\dfrac{\varphi[x/y], \Gamma \Rightarrow \Delta}{\exists_x \varphi, \Gamma \Rightarrow \Delta}\ (\mathrm{L}_\exists), \quad y \notin \mathtt{fv}(\Gamma, \Delta)$	$\dfrac{\Gamma \Rightarrow \Delta, \varphi[x/t]}{\Gamma \Rightarrow \Delta, \exists_x \varphi}\ (\mathrm{R}_\exists)$

structural rule (RC) is similar: if the set Δ, φ, φ holds from Γ then Δ, φ, obtained by removing one copy of φ in the succedent, also holds from Γ.

Example 16 To illustrate the application of the inference rules of Gentzen's sequent calculus observe a derivation of Peirce's law below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi \Rightarrow \varphi\ (\mathrm{Ax})}{\varphi \Rightarrow \varphi, \psi}\ (\mathrm{RW})}{\Rightarrow \varphi, \varphi \to \psi}\ (\mathrm{R}_\to) \qquad \varphi \Rightarrow \varphi\ (\mathrm{Ax})}{(\varphi \to \psi) \to \varphi \Rightarrow \varphi}\ (\mathrm{L}_\to)}{\Rightarrow ((\varphi \to \psi) \to \varphi) \to \varphi}\ (\mathrm{R}_\to)$$

Observe that the first application of rule (RW) can be dropped since the sequent $\varphi \Rightarrow \varphi, \psi$ is an axiom.

Example 17 As a second example, consider the following derivation of the sequent $\varphi \Rightarrow \neg\neg\varphi$, where $\neg\varphi$ is a shortcut for $\varphi \to \bot$, as previously mentioned. Notice that this sequent expresses the natural deduction derived rule $(\neg\neg_i)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\varphi \Rightarrow \varphi \ \ (\mathrm{Ax})}{\varphi \Rightarrow \varphi, \bot}\,(\mathrm{RW}) \qquad \dfrac{\bot \Rightarrow \bot \ \ (\mathrm{Ax})}{\varphi, \bot \Rightarrow \bot}\,(\mathrm{LW})}{\varphi \to \bot, \varphi \Rightarrow \bot}\,(\mathrm{L}_{\to})}{\varphi \Rightarrow (\varphi \to \bot) \to \bot}\,(\mathrm{R}_{\to})$$

As in the previous example, notice that rules (RW) and (LW) are not necessary.

Example 18 As a third example, consider the following derivation of the sequent $\neg\neg\varphi \Rightarrow \varphi$. Notice that this sequent expresses the natural deduction rule $(\neg\neg_e)$.

$$\dfrac{\dfrac{\dfrac{\varphi \Rightarrow \varphi, \bot \ \ (\mathrm{Ax})}{\Rightarrow \varphi, \varphi \to \bot}\,(\mathrm{R}_{\to}) \qquad \bot \Rightarrow \varphi \ \ (\mathrm{L}_{\bot})}{(\varphi \to \bot) \to \bot \Rightarrow \varphi}\,(\mathrm{L}_{\to})$$

Exercise 38 a. Build a derivation for *Modus Tollens*; that is, derive the sequent $\varphi \to \psi, \neg\psi \Rightarrow \neg\varphi$.

b. Build derivations for the contraposition rules, (CP_1) and (CP_2); that is, for the sequents $\varphi \to \psi \Rightarrow \neg\psi \to \neg\varphi$ and $\neg\psi \to \neg\varphi \Rightarrow \varphi \to \psi$.

c. Build derivations for the contraposition rules, (CP_3) and (CP_4).

An important observation is that weakening rules are unnecessary. Informally, the possibility of eliminating weakening rules in a derivation is justified by the fact that it would be enough to include the necessary formulas in the context just when *weakened* axioms are allowed, as in our case. When weakening rules are allowed, we only just need *non-weakened* axioms of the form "$\varphi \Rightarrow \varphi(\mathrm{Ax})$" and "$\bot \Rightarrow (\mathrm{L}_{\bot})$", which is not the case of our calculus. For instance, observe below a derivation for the sequent $\varphi \Rightarrow \neg\neg\varphi$ without applications of weakening rules.

$$\dfrac{\dfrac{\dfrac{\varphi \Rightarrow \varphi, \bot \ \ (\mathrm{Ax}) \qquad \varphi, \bot \Rightarrow \bot; \ \ (\mathrm{Ax})}{\varphi \to \bot, \varphi \Rightarrow \bot}\,(\mathrm{L}_{\to})}{\varphi \Rightarrow (\varphi \to \bot) \to \bot}\,(\mathrm{R}_{\to})$$

Exercise 39 $(*)$ Prove that weakening rules are unnecessary. It should be proved that all derivations in the sequent calculus can be transformed into a derivation without applications of weakening rules.

Hint: For doing this, you will need to apply induction on the derivations analyzing the case of application of each of the rules just before a last step of weakening. For instance, consider the case of a derivation that finishes in an application of the rule (LW) after an application of the rule $(\mathrm{L}_{\to})$:

$$\dfrac{\dfrac{\dfrac{\nabla_1 \qquad \nabla_2}{\Gamma \Rightarrow \Delta, \varphi \qquad \psi, \Gamma \Rightarrow \Delta}\,(\mathrm{L}_{\to})}{\varphi \to \psi, \Gamma \Rightarrow \Delta}\,(\mathrm{LW})}{\delta, \varphi \to \psi, \Gamma \Rightarrow \Delta}$$

Thus, a new derivation in which rules (LW) and ($L_{\rightarrow}$) are interchanged can be built as shown below:

$$\dfrac{\dfrac{\begin{matrix}\nabla_1\\ \Gamma \Rightarrow \Delta, \varphi\end{matrix}}{\delta, \Gamma \Rightarrow \Delta, \varphi}\,(\text{LW}) \qquad \dfrac{\begin{matrix}\nabla_2\\ \psi, \Gamma \Rightarrow \Delta\end{matrix}}{\psi, \delta, \Gamma \Rightarrow \Delta}\,(\text{LW})}{\delta, \varphi \rightarrow \psi, \Gamma \Rightarrow \Delta}\,(\text{L}_{\rightarrow})$$

Then, by induction hypothesis one can assume the existence of derivations without applications of weakening rules, say ∇_1' and ∇_2', for the sequents $\delta, \Gamma \Rightarrow \Delta, \varphi$ and $\psi, \delta, \Gamma \Rightarrow \Delta$, respectively. Therefore a derivation without application of weakening rules of the form below would be possible.

$$\dfrac{\begin{matrix}\nabla_1'\\ \delta, \Gamma \Rightarrow \Delta, \varphi\end{matrix} \qquad \begin{matrix}\nabla_2'\\ \psi, \delta, \Gamma \Rightarrow \Delta\end{matrix}}{\delta, \varphi \rightarrow \psi, \Gamma \Rightarrow \Delta}\,(\text{L}_{\rightarrow})$$

An additional detail should be taken in consideration in the application of the induction hypothesis: since other possible applications of weakening rules might appear in the derivations ∇_1 and ∇_2, the correct procedure is starting the elimination of weakening rules from nodes in the proof-tree in which a first application of a weakening rule is done.

Although the previous rules are sufficient (even dropping the weakening ones) for deduction in the predicate calculus, a useful rule called *cut rule* can be added. Among the applications of the cut rule, its inclusion in the sequent calculus is useful for proving that natural deduction and deduction in the sequent calculus are equivalent (Table 3.3).

In the given rule (Cut), φ is the principal formula, and Γ, Γ', Δ, Δ' is the context. This is a so called *non-sharing context* version of (Cut). Also, a so called *sharing context* version of (Cut) is possible in which $\Gamma = \Gamma'$, $\Delta = \Delta'$ and the conclusion is the sequent $\Gamma \Rightarrow \Delta$.

Intuitively, the cut rule allows for inclusion of *lemmas* in proofs: whenever one knows that φ is deducible in a context Γ, Δ and, additionally, one knows that the sequent $\varphi, \Gamma' \Rightarrow \Delta'$ is provable, then one can deduce the conclusion of the cut rule (see the next examples).

Table 3.3 Cut rule

$$\dfrac{\Gamma \Rightarrow \Delta, \varphi \qquad \varphi, \Gamma' \Rightarrow \Delta'}{\Gamma\Gamma' \Rightarrow \Delta\Delta'}\,(\text{Cut})$$

Example 19 To be more illustrative, once a proof for the sequent $\Rightarrow \neg\neg(\psi \vee \neg\psi)$ is obtained, the previous proof for the sequent $\neg\neg\varphi \Rightarrow \varphi$ (see Example 18) can be used, replacing φ by $\psi \vee \neg\psi$, to conclude by application of the cut rule that $\Rightarrow \psi \vee \neg\psi$ holds:

$$\frac{\Rightarrow \neg\neg(\psi \vee \neg\psi) \qquad \neg\neg(\psi \vee \neg\psi) \Rightarrow \psi \vee \neg\psi}{\Rightarrow \psi \vee \neg\psi}\ (\text{Cut})$$

Example 20 Also, a derivation for the sequent $\Rightarrow \neg\neg(\psi \vee \neg\psi)$ can be obtained applying the (Cut) rule using the previously proved sequent $\varphi \Rightarrow \neg\neg\varphi$ (see Example 17), replacing φ by $\psi \vee \neg\psi$, and the sequent $\Rightarrow \varphi \vee \neg\varphi$:

$$\frac{\Rightarrow \psi \vee \neg\psi \qquad \psi \vee \neg\psi \Rightarrow \neg\neg(\psi \vee \neg\psi)}{\Rightarrow \neg\neg(\psi \vee \neg\psi)}\ (\text{Cut})$$

Derivations that do not use the cut rule own an important property called the *subformula* property. Essentially, this property states that the logical rules applied in the derivation can be restricted exclusively to rules for the logical connectives that appear in the sequent in the conclusion of the derivation and, that all formulas that appear in the whole derivation are contained in the conclusion. Indeed, this property is trivially broken when the cut rule is allowed since the principal formula of an application of the cut rule does not need to belong to the conclusion of the derivation. Intuitively, the cut rule enables the use of arbitrary lemmas in the proof of a theorem.

The theorem of *cut elimination* establishes that any proof in the sequent calculus for predicate logic can be transformed in a proof without the use of the cut rule. The proof is elaborated and will not be presented here.

Theorem 12 (Cut Elimination). *Any sequent* $\Gamma \Rightarrow \Delta$ *that is provable with the sequent calculus together with the cut rule is also provable without the latter rule.*

Among the myriad applications of the subterm property and cut elimination theorem, important implications in the structure of proofs can be highlighted that would be crucial for discriminating between minimal, intuitionistic, and classical theorems as we will see in the next section. For instance, they imply the existence of a derivation of the sequent for the law of excluded middle $\Rightarrow \varphi \vee \neg\varphi$ that should use only (axioms and) logical rules for disjunction (($\mathrm{R}_\vee$) and ($\mathrm{L}_\vee$)) and for implication (($\mathrm{R}_\to$) and ($\mathrm{L}_\to$)). Thus, if one applies initially the logical rule ($\mathrm{R}_\vee$) as shown below, only two nonderivable sequents will be obtained: $\Rightarrow \varphi$ and $\Rightarrow \neg\varphi$:

$$\frac{\Rightarrow \varphi}{\Rightarrow \varphi \vee \neg\varphi}\ (\mathrm{R}_\vee) \qquad\qquad \frac{\Rightarrow \neg\varphi}{\Rightarrow \varphi \vee \neg\varphi}\ (\mathrm{R}_\vee)$$

This implies the necessity of the application of a structural rule before any application of ($\mathrm{R}_\vee$), being the unique option rule (RC):

$$\frac{\Rightarrow \varphi \vee \neg\varphi, \quad \varphi \vee \neg\varphi}{\Rightarrow \varphi \vee \neg\varphi} \text{ (RC)}$$

Exercise 40

a. Complete the derivation of the sequent for LEM: $\Rightarrow \varphi \vee \neg\varphi$.
b. Build a derivation for the sequent $\Rightarrow \neg\neg(\varphi \vee \neg\varphi)$ using neither rule (Cut) nor rule (RC).

As in natural deduction, we will use notation $\vdash \Gamma \Rightarrow \Delta$ meaning that the sequent $\Gamma \Rightarrow \Delta$ is derivable with Gentzen's sequent calculus. To discriminate we will use subscripts: $\vdash_N, \vdash_G$ and $\vdash_{G+cut}$ to denote respectively derivation by natural deduction, deduction *à la* Gentzen, and deduction *à la* Gentzen using also the cut rule. Using this notation, the cut elimination theorem can be shortly written as below:

$$\vdash_{G+cut} \Gamma \Rightarrow \Delta \quad \text{iff} \quad \vdash_G \Gamma \Rightarrow \Delta$$

In the remaining of this chapter for the Gentzen's sequent calculus, we will understand the calculus with the cut rule.

3.3 The Intuitionistic Gentzen's Sequent Calculus

As for natural deduction, it is also possible to obtain a restricted set of rules for the intuitionistic logic. It is only necessary to restrict all Gentzen's rules in Tables 3.1 and 3.2 to deal only with sequents with at most one formula in their succedents. For the minimal logic, all sequents in a derivation should have exactly one formula in their succedents. Thus, the rule (RC) should be dropped from the intuitionistic set of Gentzen's rules and, in the intuitionistic case, but not in the minimal one, the rule (RW) might be applied only to sequents with empty succedent:

$$\frac{\Gamma \Rightarrow}{\Gamma \Rightarrow \varphi} \text{ (RW)}$$

Essentially, all occurrences of Δ in Tables 3.1 and 3.2 should be adequately replaced by either none or a unique formula, say δ, except for rule (RC) that should be dropped and rule ($L_\rightarrow$) that should be changed into the specialized rule in Table 3.4.

Also, a special version of the cut rule is required as given in Table 3.5.

Example 21 Observe the derivation below for the sequent $\Rightarrow \neg\neg\varphi \rightarrow \varphi$ that is related with the nonintuitionistic property of elimination of the double negation:

Table 3.4 Left implication rule $(L_{\to})$ for the intuitionistic SC

$$\frac{\Gamma \Rightarrow \varphi \qquad \psi, \Gamma \Rightarrow \delta}{\varphi \to \psi, \Gamma \Rightarrow \delta}\ (L_{\to})$$

Table 3.5 Rule (Cut) for the intuitionistic SC

$$\frac{\Gamma \Rightarrow \varphi \qquad \varphi, \Gamma' \Rightarrow \delta}{\Gamma\Gamma' \Rightarrow \delta}\ (\text{Cut})$$

$$(R_{\to})\ \frac{\dfrac{\dfrac{\varphi \Rightarrow \varphi, \bot\ (\text{Ax})}{\Rightarrow \varphi, \neg\varphi} \qquad \dfrac{}{\bot \Rightarrow \varphi, \varphi\ (L_{\bot})}}{\neg\neg\varphi \Rightarrow \varphi, \varphi}\ (L_{\to})}{\neg\neg\varphi \Rightarrow \varphi}\ (RC)}{\Rightarrow \neg\neg\varphi \to \varphi}\ (R_{\to})$$

Since we know that this property is not intuitionistic, there would not be possible derivation of this sequent with the intuitionistic Gentzen's rules; that means any possible derivation of this sequent will include a sequent with a succedent with more than one formula (Cf. Example 18).

Observe that the same happens for the sequent $\Rightarrow \varphi \vee \neg\varphi$ (Cf. Exercise 40).

Exercise 41 (Cf. Exercise 40). Build a minimal derivation in the sequent calculus for the sequent $\Rightarrow \neg\neg(\varphi \vee \neg\varphi)$.

Observe that derivations for the sequents for *Modus Tollens* in Exercise 38 can be built in the intuitionistic Gentzen's calculus as well as for the sequent for (CP_1), but not for (CP_2).

Exercise 42 (Cf. Exercise 38). Give either intuitionistic or classical proofs *à la* Gentzen for all Gentzen's versions of (CP) according to your answers to Exercises 9 and 10.

a. $\varphi \to \psi \Rightarrow \neg\psi \to \neg\varphi$ (CP_1);
b. $\neg\varphi \to \neg\psi \Rightarrow \psi \to \varphi$ (CP_2);
c. $\neg\varphi \to \psi \Rightarrow \neg\psi \to \varphi$ (CP_3); and
d. $\varphi \to \neg\psi \Rightarrow \psi \to \neg\varphi$ (CP_4).

Exercise 43 (Cf. Exercise 38). Also, provide intuitionistic or classical derivations for the versions below of *Modus Tollens*, according to your classification in Exercise 11.

a. $\varphi \to \psi, \neg\psi \Rightarrow \neg\varphi$ (MT_1);
b. $\varphi \to \neg\psi, \psi \Rightarrow \neg\varphi$ (MT_2);
c. $\neg\varphi \to \psi, \neg\psi \Rightarrow \varphi$ (MT_3); and
d. $\neg\varphi \to \neg\psi, \psi \Rightarrow \neg\varphi$ (MT_4).

Example 22 (Cf. Example 18). Consider the following classical derivation of the sequent $\Rightarrow \forall_x(\neg\neg\varphi \to \varphi)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\varphi \Rightarrow \varphi, \bot\ (\mathrm{Ax})}{\Rightarrow \varphi, \neg\varphi}(\mathrm{R}_\to) \qquad \bot \Rightarrow \varphi\ (\mathrm{L}_\bot)}{\neg\neg\varphi \Rightarrow \varphi}(\mathrm{L}_\to)}{\Rightarrow \neg\neg\varphi \to \varphi}(\mathrm{R}_\to)}{\Rightarrow \forall_x(\neg\neg\varphi \to \varphi)}(\mathrm{R}_\forall)$$

Sequents of the form $\Rightarrow \forall(\neg\neg\varphi \to \varphi)$ are called *stability axioms* and are derivable in the strict classical calculus. There is no possible intuitionistic derivation for this kind of sequent. In fact, the reader can notice that this is related with the strictly classical rule ($\neg\neg_e$) in deduction natural. Also, the reader can check that the use of the classical rule ($\mathrm{L}_\to$) as well as the inclusion of sequents with more than one formula in the succedent are obligatory to build a derivation for this kind of sequents.

Exercise 44

1. Build an intuitionistic derivation for the sequent $\Rightarrow \neg\neg(\neg\neg\varphi \to \varphi)$.
2. Build a nonclassical derivation for the double negation of Peirce's law: $\Rightarrow \neg\neg(((\varphi \to \psi) \to \varphi) \to \varphi)$.

Exercise 45 (Cf. Exercise 12). Using the intuitionistic Gentzen's calculus build derivations for the following sequents.

a. $\neg\neg\neg\phi \Rightarrow \neg\phi$ and $\neg\phi \Rightarrow \neg\neg\neg\phi$
b. $\neg\neg(\phi \to \psi) \Rightarrow (\neg\neg\phi \to \neg\neg\psi)$.
c. $\neg\neg(\phi \wedge \psi) \Rightarrow (\neg\neg\phi \wedge \neg\neg\psi)$.
d. $\neg(\phi \vee \psi) \Rightarrow (\neg\phi \wedge \neg\psi)$ and $(\neg\phi \wedge \neg\psi) \Rightarrow \neg(\phi \vee \psi)$.

3.4 Natural Deduction Versus Deduction *à la* Gentzen

In this section, we prove that both natural deduction and deduction *à la* Gentzen have the same expressive power that means we can prove exactly the same set of theorems using natural deduction or using deduction *à la* Gentzen. Initially, we prove that the

property holds restricted to the intuitionistic logic. Then, we prove that it holds for the logic of predicates.

The main result is stated as

$$\vdash_G \Gamma \Rightarrow \varphi \quad \text{if and only if} \quad \Gamma \vdash_N \varphi$$

For proving this result, we will use an informal style of discussion which requires a deal of additional effort of the reader in order to interpret a few points that would not be presented in detail. Among others points, notice for instance that the antecedent "Γ" of the sequent $\Gamma \Rightarrow \varphi$ is in fact a *multiset* of formulas, while "Γ" as premise of $\Gamma \vdash_N \varphi$ should be interpreted as a *finite subset* of assumptions built from Γ that can be used in a natural derivation of φ.

Notice also, that in the classical sequent calculus one can build derivations for sequents of the form $\Gamma \Rightarrow \Delta$, and in natural deduction only derivations of a formula, say δ, are allowed, that is derivations of the form $\Gamma' \vdash_N \delta$. Then for the classical logic it would be necessary to establish a correspondence between derivability of arbitrary sequents of the form $\Gamma \Rightarrow \Delta$ and derivability of "equivalent" sequents with exactly one formula in the succedent of the form $\Gamma' \Rightarrow \delta$.

3.4.1 *Equivalence Between ND and Gentzen's SC—The Intuitionistic Case*

The equivalence for the case of the intuitionistic logic is established in the next theorem.

Theorem 13 (ND versus SC for the intuitionistic logic). *The equivalence below holds for the intuitionistic sequent calculus and the intuitionistic natural deduction:*

$$\vdash_G \Gamma \Rightarrow \varphi \textit{ if and only if } \Gamma \vdash_N \varphi$$

Proof According to previous observations, it is possible to consider the calculus *à la* Gentzen without weakening rules. We will prove that the intuitionistic Gentzen's sequent calculus, including the cut rule, is equivalent to intuitionistic natural deduction. The proof is by induction on the structure of derivations.

Initially, we prove necessity that is $\vdash_G \Gamma \Rightarrow \varphi$ implies $\Gamma \vdash_N \varphi$. This is done by induction on derivations in the intuitionistic Gentzen's sequent calculus, analyzing different cases according to the last rule applied in a derivation.

IB. The simplest derivations *à la* Gentzen are given by applications of rules (Ax) and ($L_\perp$):

$$\Gamma, \varphi \Rightarrow \varphi (\text{Ax}) \qquad \Gamma, \perp \Rightarrow \varphi (L_\perp)$$

In natural deduction, these proofs correspond respectively to derivations:

$$[\varphi]^u\,(\mathrm{Ax}) \qquad \frac{[\bot]^u}{\varphi}\ (\bot_e)$$

Notice that this means $\Gamma, \varphi \vdash_N \varphi$ and $\Gamma, \bot \vdash_N \varphi$, since the assumption of the former derivation φ belongs to $\Gamma \cup \{\varphi\}$ and the assumption of the latter derivation $\bot$ belongs to $\Gamma \cup \{\bot\}$.

IS. We will consider derivations in the Gentzen calculus analyzing cases according to the last rule applied in the derivation. Right rules correspond to introduction rules, and left rules will need a more elaborated analysis. First, observe that in the intuitionistic case the sole contraction rule to be considered is (LC):

$$\frac{\begin{array}{c}\nabla\\ \psi, \psi, \Gamma \Rightarrow \varphi\end{array}}{\psi, \Gamma \Rightarrow \varphi}\ (\mathrm{LC})$$

And, whenever we have a derivation finishing in an application of this rule, by induction hypothesis, there is a natural derivation of its premise $\{\psi\} \cup \{\psi\} \cup \Gamma \vdash_N \varphi$, which corresponds to $\{\psi\} \cup \Gamma \vdash_N \varphi$ because the premises in natural deduction are sets.

Case ($\mathrm{L}_\wedge$). Suppose one has a derivation of the form

$$\frac{\begin{array}{c}\nabla\\ \psi, \Gamma \Rightarrow \varphi\end{array}}{\psi \wedge \delta, \Gamma \Rightarrow \varphi}\ (\mathrm{L}_\wedge)$$

By induction hypothesis, one has a derivation for $\Gamma, \psi \vdash_N \varphi$, say ∇', whose assumptions are ψ and a finite subset Γ' of Γ. Thus a natural derivation is obtained as shown below, by replacing each occurrence of the assumption $[\psi]$ in ∇' by an application of rule ($\wedge_e$).

$$\begin{array}{c}\dfrac{[\psi \wedge \delta]^u}{\psi}\ (\wedge_e)\\ \nabla'\\ \varphi\end{array}$$

By brevity, in the previous derivation assumptions in Γ' were dropped, as will be done in all other derivations in this proof.

Case ($\mathrm{R}_\wedge$). Suppose $\varphi = \delta \wedge \psi$ and one has a derivation of the form

$$\frac{\begin{array}{c}\nabla_1\\ \Gamma \Rightarrow \delta\end{array} \qquad \begin{array}{c}\nabla_2\\ \Gamma \Rightarrow \psi\end{array}}{\Gamma \Rightarrow \delta \wedge \psi}\ (\mathrm{R}_\wedge)$$

By induction hypothesis, one has derivations for $\Gamma \vdash_N \delta$ and $\Gamma \vdash_N \psi$, say ∇'_1 and ∇'_2. Thus, a natural derivation is built from these derivations applying the rule $(\wedge_i)$ as shown below.

$$\dfrac{\begin{array}{cc}\nabla'_1 & \nabla'_2\\ \delta & \psi\end{array}}{\delta \wedge \psi}\ (\wedge_i)$$

Case $(\mathrm{L}_\vee)$. Suppose one has a derivation of the form

$$\dfrac{\begin{array}{cc}\nabla_1 & \nabla_2\\ \delta, \Gamma \Rightarrow \varphi & \psi, \Gamma \Rightarrow \varphi\end{array}}{\delta \vee \psi, \Gamma \Rightarrow \varphi}\ (\mathrm{L}_\vee)$$

By induction hypothesis, one has derivations ∇'_1 and ∇'_2 for $\delta, \Gamma \vdash_N \varphi$ and $\psi, \Gamma \vdash_N \varphi$. Thus, a natural derivation, that assumes $\delta \vee \psi$, is obtained from these derivations applying the rule $(\vee_e)$ as shown below.

$$\dfrac{\begin{array}{ccc}[\delta \vee \psi]^u & \begin{array}{c}[\delta]^v\\ \nabla'_1\\ \varphi\end{array} & \begin{array}{c}[\psi]^w\\ \nabla'_2\\ \varphi\end{array}\end{array}}{\varphi}\ (\vee_e)\ v, w$$

Case $(\mathrm{R}_\vee)$. Suppose $\varphi = \delta \vee \psi$ and one has a derivation of the form

$$\dfrac{\begin{array}{c}\nabla\\ \Gamma \Rightarrow \delta\end{array}}{\Gamma \Rightarrow \delta \vee \psi}\ (\mathrm{R}_\vee)$$

By induction hypotheses there exists a natural derivation ∇' for $\Gamma \vdash_N \delta$. Applying at the end of this derivation rule $(\vee_i)$, one obtains a natural derivation for $\Gamma \vdash_N \delta \vee \psi$.

Case $(\mathrm{L}_\to)$. Suppose one has a derivation of the form

$$\dfrac{\begin{array}{cc}\nabla_1 & \nabla_2\\ \Gamma \Rightarrow \delta & \psi, \Gamma \Rightarrow \varphi\end{array}}{\delta \to \psi, \Gamma \Rightarrow \varphi}\ (\mathrm{L}_\to)$$

By induction hypothesis there exist natural derivations ∇'_1 and ∇'_2 for $\Gamma \vdash_N \delta$ and $\psi, \Gamma \vdash_N \varphi$. A natural derivation for $\Gamma \vdash_N \varphi$ is obtained from these derivations, by replacing each assumption $[\psi]^u$ in ∇'_2 by a derivation of ψ finishing in an application of rule $(\to_e)$ with premises $[\delta \to \psi]^v$ and δ. The former as a new assumption and the latter is derived as in ∇'_1.

$$\dfrac{\dfrac{[\delta \to \psi]^v \qquad \begin{matrix}\nabla_1' \\ \delta\end{matrix}}{\psi}\ (\to_e)}{\begin{matrix}\nabla_2' \\ \varphi\end{matrix}}$$

Case ($\mathrm{R}_\to$). Suppose $\varphi = \delta \to \psi$ and one has a derivation of the form

$$\dfrac{\begin{matrix}\nabla \\ \delta, \Gamma \Rightarrow \psi\end{matrix}}{\Gamma \Rightarrow \delta \to \psi}\ (\mathrm{R}_\to)$$

By induction hypothesis, there exists a natural derivation ∇' for $\delta, \Gamma \vdash_N \psi$. The natural derivation for $\Gamma \vdash_N \delta \to \psi$ is obtained by applying at the end of this proof rule $(\to_i)$ discharging assumptions $[\delta]^u$ as depicted below.

$$\dfrac{\begin{matrix}[\delta]^u \\ \nabla' \\ \psi\end{matrix}}{\delta \to \psi}\ (\to_i)\ u$$

Case ($\mathrm{L}_\forall$). Suppose one has a derivation of the form

$$\dfrac{\begin{matrix}\nabla \\ \psi[x/y], \Gamma \Rightarrow \varphi\end{matrix}}{\forall_x \psi, \Gamma \Rightarrow \varphi}\ (\mathrm{L}_\forall)$$

Then by induction hypothesis there exists a natural derivation for $\psi[x/y], \Gamma \vdash_N \varphi$, say ∇'. A natural derivation for $\forall_x \psi, \Gamma \vdash_N \varphi$ is obtained by replacing all assumptions of $[\psi[x/y]]^u$ in ∇' by a deduction of $\psi[x/y]$ with assumption $[\forall_x \psi]^v$ applying rule $(\forall_e)$.

Case ($\mathrm{R}_\forall$). Suppose $\varphi = \forall_x \psi$ and one has a derivation of the form

$$\dfrac{\begin{matrix}\nabla \\ \Gamma \Rightarrow \psi[x/y]\end{matrix}}{\Gamma \Rightarrow \forall_x \psi}\ (\mathrm{R}_\forall)$$

where $y \notin \mathtt{fv}(\Gamma)$. Then by induction hypothesis there exists a natural derivation ∇' for $\Gamma \vdash_N \psi[x/y]$. Thus a simple application at the end of ∇' of rule $(\forall_i)$, that is possible since y does not appear in the open assumptions, will complete the desired natural derivation.

Case ($L_\exists$). Suppose one has a derivation of the form

$$\dfrac{\begin{array}{c}\nabla\\ \psi[x/y], \Gamma \Rightarrow \varphi\end{array}}{\exists_x \psi, \Gamma \Rightarrow \varphi}\ (L_\exists)$$

where $y \notin \mathtt{fv}(\Gamma, \varphi)$. By induction hypothesis there exists a natural derivation ∇' for $\psi[x/y], \Gamma \vdash_N \varphi$. The desired derivation is built by an application of rule $(\exists_e)$ $u$sing as premises the assumption $[\exists_x \psi]^v$ and the conclusion of ∇'. In this application assumptions of $[\psi[x/y]]^u$ in ∇' are discharged as depicted below. Notice that the application of rule $(\exists_e)$ is possible since $y \notin \mathtt{fv}(\Gamma, \varphi)$, which implies it does not will appear in open assumptions in ∇'.

$$\dfrac{[\exists_x \psi]^v \qquad \begin{array}{c}[\psi[x/y]]^u\\ \nabla'\\ \varphi\end{array}}{\varphi}\ (\exists_e)\ u$$

Case ($R_\exists$). Suppose $\varphi = \exists_x \psi$ and one has a derivation of the form

$$\dfrac{\begin{array}{c}\nabla\\ \Gamma \Rightarrow \psi[x/t]\end{array}}{\Gamma \Rightarrow \exists_x \psi}\ (R_\exists)$$

A natural derivation for $\Gamma \vdash_N \exists_x \psi$ is built by induction hypothesis which gives a natural derivation ∇' for $\Gamma \vdash_N \psi[x/t]$ and application of rule $(\exists_i)$ to the conclusion of ∇'.

Case (cut). Suppose one has a derivation finishing in an application of rule (Cut) as shown below

$$\dfrac{\begin{array}{c}\nabla_1\\ \Gamma \Rightarrow \psi\end{array} \qquad \begin{array}{c}\nabla_2\\ \psi, \Gamma \Rightarrow \varphi\end{array}}{\Gamma \Rightarrow \varphi}\ (\text{Cut})$$

By induction hypothesis there are natural derivations ∇_1' and ∇_2' for $\Gamma \vdash_N \psi$ and $\psi, \Gamma \vdash_N \varphi$. To obtain the desired natural derivation, all assumptions $[\psi]^u$ in ∇_2' are replaced by derivations of ψ using ∇_1':

$$\begin{array}{c}\nabla_1'\\ \psi\\ \nabla_2'\\ \varphi\end{array}$$

Now we prove sufficiency that is $\vdash_G \Gamma \Rightarrow \varphi$ whenever $\Gamma \vdash_N \varphi$. The proof is by induction on the structure of natural derivations analyzing the last applied rule.

IB. Proofs consisting of a sole node $[\varphi]^u$ correspond to applications of (Ax): $\Gamma \Rightarrow \varphi$, where $\varphi \in \Gamma$.

IS. All derivations finishing in introduction rules are straightforwardly related with derivations *à la* Gentzen finishing in the corresponding right rule as in the proof of necessity. Only one example is given: $(\rightarrow_i)$. The other cases are left as an exercise for the reader.

Suppose $\varphi = \delta \rightarrow \psi$ and one has a derivation finishing in an application of $(\rightarrow_i)$ discharging assumptions of δ and using assumptions in Γ:

$$\dfrac{\begin{array}{c}[\delta]^u\\ \nabla\\ \psi\end{array}}{\delta \rightarrow \psi}\ (\rightarrow_i)\ u$$

By induction hypothesis there exists a derivation *à la* Gentzen ∇' for the sequent $\delta, \Gamma \Rightarrow \psi$. Thus, the desired derivation is built by a simple application of rule $(\mathrm{R}_\rightarrow)$:

$$\dfrac{\begin{array}{c}\nabla'\\ \delta, \Gamma \Rightarrow \psi\end{array}}{\Gamma \Rightarrow \delta \rightarrow \psi}\ (\mathrm{R}_\rightarrow)$$

Derivations finishing in elimination rules will require application of the rule (Cut). A few interesting cases are given. All the other cases remain as an exercise for the reader.

Case $(\vee_e)$. Suppose one has a natural derivation for $\Gamma \vdash_N \varphi$ finishing in an application of rule $(\vee_e)$ as shown below.

$$\dfrac{\begin{array}{ccc}\nabla & [\delta]^v & [\psi]^w\\ \delta \vee \psi & \nabla_1 & \nabla_2\\ & \varphi & \varphi\end{array}}{\varphi}\ (\vee_e)\ v, w$$

By induction hypothesis, there are derivations *à la* Gentzen ∇', ∇'_1 and ∇'_2 respectively, for the sequents $\Gamma \Rightarrow \delta \vee \psi$, $\delta, \Gamma \Rightarrow \varphi$ and $\psi, \Gamma \Rightarrow \varphi$. Thus, using these derivations, a derivation for $\Gamma \Rightarrow \varphi$ is built as shown below.

$$\dfrac{\begin{array}{cc}\nabla' & \dfrac{\begin{array}{cc}\nabla'_1 & \nabla'_2\\ \delta, \Gamma \Rightarrow \varphi & \psi, \Gamma \Rightarrow \varphi\end{array}}{\delta \vee \psi, \Gamma \Rightarrow \varphi}\ (\mathrm{L}_\vee)\\ \Gamma \Rightarrow \delta \vee \psi & \end{array}}{\Gamma \Rightarrow \varphi}\ (\mathrm{Cut})$$

Case $(\rightarrow_e)$. Suppose one has a natural derivation for $\Gamma \vdash_N \varphi$ that finishes in an application of $(\rightarrow_e)$ as shown below.

$$\dfrac{\begin{array}{cc}\nabla_1 & \nabla_2\\ \delta & \delta \rightarrow \varphi\end{array}}{\varphi}\ (\rightarrow_e)$$

By induction hypothesis, there are derivations *à la* Gentzen ∇_1' and ∇_2' for the sequents $\Gamma \Rightarrow \delta$ and $\Gamma \Rightarrow \delta \to \varphi$, respectively. The desired derivation is built, using these derivations, as depicted below.

$$\dfrac{\nabla_2' \quad \Gamma \Rightarrow \delta \to \varphi \qquad \dfrac{\nabla_1' \quad \Gamma \Rightarrow \delta \qquad \varphi, \Gamma \Rightarrow \varphi \ (\mathrm{Ax})}{\delta \to \varphi, \Gamma \Rightarrow \varphi}\ (\mathrm{L}_{\to})}{\Gamma \Rightarrow \varphi}\ (\mathrm{Cut})$$

Case $(\exists_e)$. Suppose one has a natural derivation for $\Gamma \vdash_N \varphi$ finishing in an application of the rule $(\exists_e)$ as shown below.

$$\dfrac{\nabla_1 \quad [\exists_x \psi]^v \qquad \begin{matrix}[\psi[x/y]]^u \\ \nabla_2 \\ \varphi\end{matrix}}{\varphi}\ (\exists_e)\ u$$

By induction hypothesis, there are derivations *à la* Gentzen ∇_1' and ∇_2' for the sequents $\Gamma \Rightarrow \exists_x \psi$ and $\psi[x/y], \Gamma \Rightarrow \varphi$, respectively. The derivation is built as shown below. Notice that $y \notin \mathtt{fv}(\Gamma, \varphi)$, which allows the application of the rule $(\mathrm{L}_{\exists})$.

$$\dfrac{\nabla_1' \quad \Gamma \Rightarrow \exists_x \psi \qquad \dfrac{\nabla_2' \quad \psi[x/y], \Gamma \Rightarrow \varphi}{\exists_x \psi, \Gamma \Rightarrow \varphi}\ (\mathrm{L}_{\exists})}{\Gamma \Rightarrow \varphi}\ (\mathrm{Cut})$$

□

Exercise 46 Prove all remaining cases in the proof of sufficiency of Theorem 13.

3.4.2 *Equivalence of ND and Gentzen's SC—The Classical Case*

Before proving equivalence of natural deduction and deduction *à la* Gentzen for predicate logic, a few additional definitions and properties are necessary. First of all, we define a notion that makes it possible to transform any sequent in an equivalent one but with only one formula in its succedent.

By $\overline{\Gamma}$ we generically denote any sequence of formulas built from the formulas in the sequence Γ, replacing each formula in Γ by either its negation or, when the head symbol of the formula is the negation symbol, eliminating it from the formula. For instance, let $\Delta = \delta_1, \neg\delta_2, \neg\delta_3, \delta_4$, then $\overline{\Delta}$ might represent sequences as $\neg\delta_1, \neg\neg\delta_2, \delta_3, \neg\delta_4$; $\neg\delta_1, \delta_2, \delta_3, \neg\delta_4$, etc. This transformation is not only relevant

for our purposes in this chapter, but also in computational frameworks, as we will see in the next chapter, in order to get rid automatically of negative formulas in sequents that appear in a derivation.

Definition 30 (c-equivalent sequents). We will say that sequents $\varphi, \Gamma \Rightarrow \Delta$ and $\Gamma \Rightarrow \Delta, \neg\varphi$ as well as $\Gamma \Rightarrow \Delta, \varphi$ and $\neg\varphi, \Gamma \Rightarrow \Delta$ are c-equivalent in one step. The equivalence closure of this relation is called the c-equivalence relation on sequents and is denoted as $\equiv_{ce}$.

According to the previous notational convention, $\Gamma, \Gamma' \Rightarrow \Delta, \Delta'$ and $\Gamma, \overline{\Delta'} \Rightarrow \Delta, \overline{\Gamma'}$ are c-equivalent; that is,

$$\Gamma, \Gamma' \Rightarrow \Delta, \Delta' \quad \equiv_{ce} \quad \Gamma, \overline{\Delta'} \Rightarrow \Delta, \overline{\Gamma'}$$

Lemma 8 (One-step c-equivalence) *The following properties hold in the sequent calculus à la Gentzen for the classical logic:*

(i) There exists a derivation for $\vdash_G \varphi, \Gamma \Rightarrow \Delta$*, if and only if there exists a derivation for* $\vdash_G \Gamma \Rightarrow \Delta, \neg\varphi$.

(ii) There is a derivation for $\vdash_G \neg\varphi, \Gamma \Rightarrow \Delta$*, if and only if there is a derivation for* $\vdash_G \Gamma \Rightarrow \Delta, \varphi$.

Proof We consider the derivations below.

(i) **Necessity**: Let ∇ be a derivation for $\vdash_G \varphi, \Gamma \Rightarrow \Delta$. Then the desired derivation is built as follows:

$$\dfrac{\dfrac{\dfrac{\nabla}{\varphi, \Gamma \Rightarrow \Delta}}{\varphi, \Gamma \Rightarrow \Delta, \bot}\,(\mathrm{RW})}{\Gamma \Rightarrow \Delta, \neg\varphi}\,(\mathrm{R}_{\to})$$

Sufficiency: Let ∇ be a derivation for $\vdash_G \Gamma \Rightarrow \Delta, \neg\varphi$. Then the desired derivation is built as follows:

$$\dfrac{\dfrac{\dfrac{\nabla}{\Gamma \Rightarrow \Delta, \neg\varphi}}{\varphi, \Gamma \Rightarrow \Delta, \neg\varphi}\,(\mathrm{LW}) \qquad \dfrac{(\mathrm{Ax})\ \varphi, \Gamma \Rightarrow \Delta, \varphi \qquad \bot, \varphi, \Gamma \Rightarrow \Delta\ (\mathrm{L}_{\bot})}{\neg\varphi, \varphi, \Gamma \Rightarrow \Delta}\,(\mathrm{L}_{\to})}{\varphi, \Gamma \Rightarrow \Delta}\,(\mathrm{Cut})$$

Observe that in both cases, when Δ is the empty sequence we have an intuitionistic proof.

(ii) **Necessity**: Let ∇ be a derivation for $\vdash_G \neg\varphi, \Gamma \Rightarrow \Delta$. Then the desired derivation is built as follows:

$$\dfrac{\Gamma \Rightarrow \Delta, \varphi, \neg\neg\varphi \to \varphi \qquad \dfrac{\dfrac{\dfrac{\dfrac{\nabla}{\neg\varphi, \Gamma \Rightarrow \Delta}}{\neg\varphi, \Gamma \Rightarrow \Delta, \varphi, \bot}\,(\mathrm{RW})}{\Gamma \Rightarrow \Delta, \varphi, \neg\neg\varphi}\,(\mathrm{R}_{\to}) \qquad \varphi, \Gamma \Rightarrow \Delta, \varphi\ (\mathrm{Ax})}{\neg\neg\varphi \to \varphi, \Gamma \Rightarrow \Delta, \varphi}\,(\mathrm{L}_{\to})}{\Gamma \Rightarrow \Delta, \varphi}\,(\mathrm{Cut})$$

where ∇' is the derivation below:

$$\dfrac{\dfrac{\dfrac{\varphi, \Gamma \Rightarrow \Delta, \varphi, \varphi, \bot\ (\mathrm{Ax})}{\Gamma \Rightarrow \Delta, \varphi, \varphi, \neg\varphi}\,(\mathrm{R}_{\to}) \qquad \bot, \Gamma \Rightarrow \Delta, \varphi, \varphi\ (\mathrm{L}_{\bot})}{\neg\neg\varphi, \Gamma \Rightarrow \Delta, \varphi, \varphi}\,(\mathrm{L}_{\to})}{\Gamma \Rightarrow \Delta, \varphi, \neg\neg\varphi \to \varphi}\,(\mathrm{R}_{\to})$$

Observe that this case is strictly classic because the left premise of (Cut), that is the derivation ∇', is essentially a proof of the sequent $\Rightarrow \neg\neg\varphi \to \varphi$ (Also, see Examples 18 and 22).

Sufficiency: Let ∇ be a derivation for $\vdash_G \Gamma \Rightarrow \Delta, \varphi$. Then the desired derivation is built as follows:

$$\dfrac{\dfrac{\nabla}{\Gamma \Rightarrow \Delta, \varphi} \qquad \bot, \Gamma \Rightarrow \Delta\ (\mathrm{L}_{\bot})}{\neg\varphi, \Gamma \Rightarrow \Delta}\,(\mathrm{L}_{\to})$$

Observe that in this case, when Δ is the empty sequence we have an intuitionistic proof.

□

Corollary 3 (One-step c-equivalence in the intuitionistic calculus) *The following properties hold in the* **intuitionistic** *calculus à la Gentzen:*

(i) There is a derivation for $\vdash_G \varphi, \Gamma \Rightarrow$*, if and only if there is a derivation for* $\vdash_G \Gamma \Rightarrow \neg\varphi$.

(ii) Assuming that $\Rightarrow \neg\neg\varphi \to \varphi$*, the existence of a derivation for* $\vdash_G \neg\varphi, \Gamma \Rightarrow$*, implies the existence of a derivation for* $\vdash_G \Gamma \Rightarrow \varphi$.

(iii) There exists a derivation for $\vdash_G \neg\varphi, \Gamma \Rightarrow$*, whenever there is a derivation for* $\vdash_G \Gamma \Rightarrow \varphi$.

Proof The proof is obtained from the proof of Lemma 8, according to the observations given in that proof. In particular for the item ii), the proof of sufficiency of the lemma is easily modified as shown below.

$$\dfrac{(\mathrm{RW})\ \dfrac{(\text{Assumption})\quad \Rightarrow \neg\neg\varphi \to \varphi}{\Gamma \Rightarrow \neg\neg\varphi \to \varphi} \qquad \dfrac{\dfrac{\dfrac{\neg\varphi, \Gamma \Rightarrow}{\neg\varphi, \Gamma \Rightarrow \bot}\,(\mathrm{RW})}{\Gamma \Rightarrow \neg\neg\varphi}\,(\mathrm{R}_{\to}) \qquad \varphi, \Gamma \Rightarrow \varphi\ (\mathrm{Ax})}{\neg\neg\varphi \to \varphi, \Gamma \Rightarrow \varphi}\,(\mathrm{L}_{\to})}{\Gamma \Rightarrow \varphi}\,(\mathrm{Cut})$$

□

Exercise 47 Complete the proof of the Corollary 3.

Lemma 9 *(c*-equivalence*) Let* $\Gamma \Rightarrow \Delta$ *and* $\Gamma' \Rightarrow \Delta'$ *be c-equivalent sequents that is* $\Gamma \Rightarrow \Delta \equiv_{ce} \Gamma' \Rightarrow \Delta'$. *Then the following holds in the* **classical** *Gentzen's sequent calculus:*

$$\vdash_G \Gamma \Rightarrow \Delta \text{ if and only if } \vdash_G \Gamma' \Rightarrow \Delta'$$

Proof (Sketch) Suppose, $\Gamma \Rightarrow \Delta$ equals $\Gamma^1, \Gamma^2 \Rightarrow \Delta^1, \Delta^2$ and $\Gamma' \Rightarrow \Delta'$ equals $\Gamma^1, \overline{\Delta^2} \Rightarrow \Delta^1, \overline{\Gamma^2}$. The proof is by induction on $n = |\Gamma^2, \Delta^2|$, that is the number of switched formulas (from the succedent to the antecedent and vice versa), that are necessary to obtain $\Gamma' \Rightarrow \Delta'$ from $\Gamma \Rightarrow \Delta$ by a number n of one-step c-equivalence transformations. Suppose $\Gamma^1, \Gamma^2_k, \overline{\Delta^2_k} \Rightarrow \Delta^1, \Delta^2_k, \overline{\Gamma^2_k}$, for $0 \leq k \leq n$, is the sequent after k one-step c-equivalence transformations, being $\Gamma^2_0 = \Gamma^2$, $\Delta^2_0 = \Delta^2$ (thus, being $\overline{\Delta^2_0}$ and $\overline{\Gamma^2_0}$ empty sequences) and Γ^2_n and Δ^2_n empty sequences (thus, being $\overline{\Gamma^2_n} = \overline{\Gamma^2}$ and $\overline{\Delta^2_n} = \overline{\Delta^2}$).

In the inductive step, for $k < n$, one assumes that there is a proof of the sequent: $\vdash_G \Gamma^1, \Gamma^2_k, \overline{\Delta^2_k} \Rightarrow \Delta^1, \Delta^2_k, \overline{\Gamma^2_k}$. Thus, applying an one-step c-equivalence transformation, by Lemma 8, one obtains a proof for $\vdash_G \Gamma^1, \Gamma^2_{k+1}, \overline{\Delta^2_{k+1}} \Rightarrow \Delta^1, \Delta^2_{k+1}, \overline{\Gamma^2_{k+1}}$. □

Exercise 48 Complete all details of the proof of Lemma 9.

In order to extend the c-equivalence Lemma from classical to intuitionistic logic, it is necessary to assume all necessary stability axioms (Cf. item 2 of Corollary 3).

Definition 31 (Intuitionistic derivability modulo stability axioms) A stability axiom is a sequent of the form $\Rightarrow \forall_x(\neg\neg\varphi \to \varphi)$. Intuitionistic derivability modulo stability axioms is defined as intuitionistic derivability assuming all possible stability axioms. Intuitionistic derivability *à la* Gentzen with stability axioms will be denoted as $\vdash_{Gi+St}$.

Lemma 10 (Equivalence between classical and intuitionistic SC modulo stability axioms) *For all sequents* $\Gamma \Rightarrow \delta$ *the following property holds:*

$$\vdash_G \Gamma \Rightarrow \delta \text{ iff } \vdash_{Gi+St} \Gamma \Rightarrow \delta$$

Therefore, for any sequent $\Gamma' \Rightarrow \Delta'$ *c-equivalent to* $\Gamma \Rightarrow \delta$, $\vdash_G \Gamma' \Rightarrow \Delta'$ *iff* $\vdash_{Gi+St} \Gamma \Rightarrow \delta$.

Proof (Sketch) To prove that $\vdash_{Gi+St} \Gamma \Rightarrow \delta$ implies $\vdash_G \Gamma \Rightarrow \delta$, suppose that ∇ is a derivation for $\vdash_{Gi+St} \Gamma \Rightarrow \delta$. The derivation ∇ is transformed into a classical derivation in the following manner: for any stability axiom assumption, that is a sequent of the form $\Rightarrow \forall_x(\neg\neg\varphi \rightarrow \varphi)$ that appears as a leaf in the derivation ∇, replace the assumption by a classical proof for $\vdash_G \Rightarrow \forall_x(\neg\neg\varphi \rightarrow \varphi)$. In this way, after all stability axiom assumptions are replaced by classical derivations, one obtains a classical derivation, say ∇', for $\vdash_G \Gamma \Rightarrow \delta$. Additionally, by Lemma 9, $\vdash_G \Gamma \Rightarrow \delta$ if and only if there exists a classical derivation for $\vdash_G \Gamma' \Rightarrow \Delta'$.

To prove that $\vdash_{Gi+St} \Gamma \Rightarrow \delta$ whenever $\vdash_G \Gamma \Rightarrow \delta$, one applies induction on the structure of the classical derivation. Most rules require a direct analysis, for instance the inductive step for rule ($R_\rightarrow$) is given below.

Case ($R_\rightarrow$). The derivation is of the form given below.

$$\dfrac{\begin{array}{c}\nabla\\ \Gamma, \varphi \Rightarrow \psi\end{array}}{\Gamma \Rightarrow' \varphi \rightarrow \psi}\ (R_\rightarrow)$$

By induction hypothesis there exists a derivation ∇' for $\vdash_{Gi+St} \Gamma, \varphi \Rightarrow \psi$, Thus, the desired derivation is obtained simply by an additional application of rule ($R_\rightarrow$)to the conclusion of the intuitionistic derivation ∇'.

The interesting case happens for rule ($L_\rightarrow$) since this rule requires two formulas in the succedent of one of the premises. The analysis of the inductive step for rule ($L_\rightarrow$) is given below.

Case ($L_\rightarrow$). The last step of the proof is of the form below, where $\Gamma = \Gamma'', \varphi \rightarrow \psi$.

$$\dfrac{\begin{array}{cc}\nabla_1 & \nabla_2\\ \Gamma'' \Rightarrow \delta, \varphi & \psi, \Gamma'' \Rightarrow \delta\end{array}}{\Gamma'', \varphi \rightarrow \psi \Rightarrow \delta}\ (L_\rightarrow)$$

By induction hypothesis there exist derivations, say ∇_1' and ∇_2', for $\vdash_{Gi+St} \Gamma'', \neg\delta \Rightarrow \varphi$ and $\vdash_{Gi+St} \psi, \Gamma, \neg\delta \Rightarrow$. Notice that the argumentation is not as straightforwardly as it appears, since it is necessary to build first classical derivations for $\vdash_G \Gamma'', \neg\delta \Rightarrow \varphi$ and $\vdash_G \psi, \Gamma'', \neg\delta \Rightarrow$ using (Lemma 8 and Corollary 3).

Thus, a derivation for $\vdash_{Gi+St} \Gamma'', \varphi \rightarrow \psi, \neg\delta \Rightarrow$ is obtained as shown below.

$$\dfrac{\begin{array}{cc}\nabla_1' & \nabla_2'\\ \Gamma'', \neg\delta \Rightarrow \varphi & \psi, \Gamma'', \neg\delta \Rightarrow\end{array}}{\Gamma'', \varphi \rightarrow \psi, \neg\delta \Rightarrow}\ (L_\rightarrow)$$

By a final application of Corollary 3 there exists a derivation for $\vdash_{Gi+St} \Gamma, \varphi \rightarrow \psi \Rightarrow \delta$. □

Exercise 49 Prove the remaining cases of the proof of Lemma 10.

Theorem 14 (Natural versus deduction *à la* Gentzen for the classical logic) *One has that for the classical Gentzen and natural calculus*

$$\vdash_G \Gamma \Rightarrow \varphi \textit{ if and only if } \Gamma \vdash_N \varphi$$

Proof (Sketch) By previous Lemma, $\vdash_G \Gamma \Rightarrow \varphi$ if and only if $\vdash_{Gi+St} \Gamma \Rightarrow \varphi$. Thus, we only require to prove that $\vdash_{Gi+St} \Gamma \Rightarrow \varphi$ if and only if $\Gamma \vdash_N \varphi$.

On the one side, an intuitionistic sequent calculus derivation modulo stability axioms for $\Gamma \Rightarrow \varphi$ will include some assumptions of the form $\Rightarrow \forall_x(\neg\neg\varphi_i \to \varphi_i)$, for formulas φ_i, with $i \leq k$ for some k in $\mathbb{N}$. Thus, by Theorem 13 there exists an intuitionistic proof in natural deduction using these stability axioms as assumptions. This intuitionistic natural derivation is converted into a classical derivation by including classical natural derivations for these assumptions.

On the other side, suppose that $\Gamma \vdash_N \varphi$ and let us assume that ∇ is a natural derivation for $\Gamma \vdash_N \varphi$ that uses only the classical rule $(\neg\neg_e)$; that is ∇ has no application of other exclusively classical rules such as (PBC) or (LEM). The derivation ∇ is transformed into an intuitionistic derivation with assumptions of stability axioms by applying to any application of the rule $(\neg\neg_e)$ in ∇, the following transformation:

$$\dfrac{\begin{array}{c}\nabla' \\ \neg\neg\varphi\end{array}}{\varphi}\,(\neg\neg_e) \quad \rightsquigarrow \quad \dfrac{\begin{array}{cc}\nabla' & \dfrac{[\forall_x(\neg\neg\varphi \to \varphi)]^u}{\neg\neg\varphi \to \varphi}\,(\forall_e) \\ \neg\neg\varphi & \end{array}}{\varphi}\,(\to_e)$$

In this manner, after replacing all applications of the rule $(\neg\neg_e)$, one obtains an intuitionistic natural derivation that has the original assumptions in Γ plus other assumptions that are stability axioms, say $\Gamma' = \forall_{x_1}(\neg\neg\varphi_1 \to \varphi_1), \ldots, \forall_{x_k}(\neg\neg\varphi_k \to \varphi_k)$, for some k in $\mathbb{N}$. By Theorem 13 there exists an intuitionistic derivation a la Gentzen, say ∇'', for $\vdash_{Gi}\ \Gamma, \Gamma' \Rightarrow \varphi$. To conclude, note that one can get rid of all formulas in Γ' by using stability axioms of the form $\Rightarrow \forall_{x_i}(\neg\neg\varphi_i \to \varphi_i)$, for $i = 1, .., k$, and applications of the (Cut) rule as depicted below.

$$\dfrac{\begin{array}{cc} & \nabla'' \\ \Rightarrow \forall_{x_1}(\neg\neg\varphi_1 \to \varphi_1) & \Gamma, \Gamma' \Rightarrow \varphi \\ \hline \Gamma, \forall_{x_2}(\neg\neg\varphi_2 \to \varphi_2), \ldots, \forall_{x_k}(\neg\neg\varphi_k \to \varphi_k) \Rightarrow \varphi & \end{array}\,(\text{Cut})}{\begin{array}{cc} \Rightarrow \forall_{x_k}(\neg\neg\varphi_k \to \varphi_k) & \Gamma, \forall_{x_k}(\neg\neg\varphi_k \to \varphi_k) \Rightarrow \varphi \\ \hline \Gamma \Rightarrow \varphi & \end{array}\,(\text{Cut})}$$

(k applications of (Cut))

This gives the desired derivation for $\vdash_{Gi+St} \Gamma \Rightarrow \varphi$. □

Exercise 50 Prove all details of Theorem 14.

Chapter 4
Derivations and Formalizations

The deductive rules studied in the previous chapters have been implemented in several computational environments such as theorem provers and proof assistants. Nowadays, one of the challenges in computer science is related with the development of tools which assist computer engineers and scientists to mathematically verify software and hardware. And this can be done in several computational tools from which here we have selected the Prototype Verification System (PVS).

Although we will explain how deductive rules can be used in this specific proof assistant, the emphasis will be on providing a general view to the reader about the mechanics of the use of the logical rules inside any deductive environment. Our selection of the proof assistant PVS is circumstantial: we have a substantial experience with this (and others) proof assistant(s) that has shown us that it is indeed an excellent tool that provides good automation mechanisms and an adequate learning curve so that after a few sections students are able to specify and formalize their own ideas. PVS provides a simple and flexible functional specification language as well as a simple interactive proof interface in which the relation between deduction rules (theory) and proof commands (practice) is apparent.

4.1 Formalizations in PVS Versus Derivations

The proof assistant PVS consists of a specification language and a proof language. The former one is used to specify functions, procedures, predicates, and logical expressions and the second one to apply deductive rules in order to prove properties of the specified predicates, functions, and procedures. Although the prover engine of PVS is more powerful than the first-order deductive systems presented in this book, our examples will be restricted to the first-order case. Namely, PVS uses a higher order specification language and a higher order proof system enriched with

M. Ayala-Rincón and F.L.C. de Moura, *Applied Logic for Computer Scientists*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-51653-0_4

a sophisticated-type theory whose deductive rules are also embedded as part of the deductive system.

The intention in this chapter is not to provide an exhaustive view of all syntactic and semantic feasibilities of PVS, but only to introduce minimal syntactic and semantic elements in order to be able to show how the proof language is related with the logical deductive mechanisms studied in the previous chapters. Full syntactical and semantic descriptions of PVS can be found in the documentation available for this proof assistant at NASA Langley PVS library http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library and at SRI International PVS site http://pvs.csl.sri.com. Among the interesting documentation, which includes up to date system guide, language reference, prover guide, etc., a good description of the type theory, deductive rules, and in general of the semantics of PVS can be found in [24], and an overview of the system in [25]. PVS formalizations related with sorting examples discussed in this and next Chap. 5 are available as the theory `sorting` in the NASA PVS library and also in the web page of the book http://logic4CS.cic.unb.br.

The style of the deductive rules of PVS is *à la* Gentzen. Thus we will relate proof commands of this system with inference rules of the Gentzen Sequent Calculus as given in the previous chapter.

4.1.1 The Syntax of the PVS Specification Language

We will restrict our attention to a subset of the specification language of PVS. The language we will use is essentially a functional language admitting recursive definitions and conditional or branching instructions such as **if then else** and binding instructions such as **let in**.

The syntax of command expressions is the following:

$$cmd \ ::= \textbf{if}\ form\ \ \textbf{then}\ cmd\ \textbf{else}\ cmd\ \ \textbf{endif}\ \mid\ \textbf{let}\ s = s\ \ \textbf{in}\ \ cmd\ \mid g(s, \ldots, s),$$

where $form$ is a conditional expression written according to the syntax of logical expressions below, s is a term, and $g(s, \ldots, s)$ is a function call.

The syntax of functional definitions is the following:

$$f(s : \tau, \ldots, s : \tau) : \ \textbf{recursive}\ \tau = \ cmd\ \ \textbf{measure}\ m(s, \ldots, s),$$

where **recursive** is obligatory when the body cmd of the definition includes calls to the function f that is being defined, and **measure** receives a function m on the parameters of the definition. The function m is used to compare the "size" of the arguments given to the function f. In the definition of functions, one distinguishes between function calls of pre-defined functions that are considered to be well-defined, and recursive calls of the same function being defined. A function p of boolean type (`bool`) is a predicate. Other basic types include `nat` and `posnat` and with these

types it is possible to build another functional types and abstract data types as for instance lists `list[T]`, finite sequences `finseq[T]`, etc. Functional types are built with the constructor `[_->_]`, so for instance

```
x : [ [list[nat]— > nat]— >bool ]
```

specifies that x is a predicate of functional type from lists of naturals to naturals.

Logical formulas are built from atomic formulas, which predicates applied to terms according to the syntax below, where $x : T$ denotes that the term variable x has type T:

$$\begin{aligned} \mathit{form} \;::=\; & p(s,\ldots,s) \mid \textbf{not}\ \mathit{form} \mid \mathit{form}\ \textbf{and}\ \mathit{form} \mid \mathit{form}\ \textbf{or}\ \mathit{form} \mid \\ & \mathit{form}\ \textbf{implies}\ \mathit{form} \mid \mathit{form}\ \textbf{iff}\ \mathit{form} \mid \\ & \textbf{if}\ \mathit{form}\ \textbf{then}\ \mathit{form}\ \textbf{else}\ \mathit{form}\ \textbf{endif} \mid \textbf{let}\ s = t\ \textbf{in}\ \mathit{form} \\ & \textbf{forall}\ (x : T) : \mathit{form} \mid \textbf{exists}\ (x : T) : \mathit{form}\ . \end{aligned}$$

The PVS syntax for the connectives **implies** and **iff** also admits the use of **=>** and **<=>**, respectively. In the sequel, instead of using this syntax we will use the standard notation for logical connectives and quantifiers.

As an initial example, we consider a PVS specification of the gcd function discussed in the introduction. The parameters m and n are of type $\mathbb{N}$ and $\mathbb{N}^+$, respectively, which in PVS are given by the basic types `posnat` and `nat`.

```
gcd(m : nat, n : posnat) : recursive nat =
if m = n then
| m
else
  if m = 0 then
  | n
  else
    if m > n then
    | gcd(m − n, n)
    else
    | gcd(n − m, m)
    end
  end
end
measure m + n
```

Algorithm 3: Specification of gcd in PVS

Two distinguishing elements should be explained here: first, the use of the keyword **recursive** indicates that the specification of gcd admits recursive calls and, second, the keyword **measure** is obligatory for recursive definitions, and should be succeeded by a measure function specified by the user, and built using the parameters of the function being defined, which are equal to m and n in the case of gcd. Below we will explain the choice of the measure function $(m, n) \mapsto m + n$.

Automatically, during type checking a specification, PVS will generate *Type Correctness Conditions*, for brevity we will write TCCs, related with the well-definedness of the functions being specified.

In the case of the function gcd, TCCs that guarantee the preservation of types for the arguments of the recursive calls are generated. For the first and second recursive calls of gcd, that are "$\gcd(m-n,n)$" and "$\gcd(n-m,m)$," respectively, these TCCs express that, under the conditions in which each one of these calls is executed, the first and second arguments are, respectively, a natural and a positive natural, as listed below:

$$\forall(m:\mathbb{N},n:\mathbb{N}^+):(m\neq n\wedge m\neq 0\wedge m>n)\rightarrow(m-n\geq 0\wedge n>0)$$

$$\forall(m:\mathbb{N},n:\mathbb{N}^+):(m\neq n\wedge m\neq 0\wedge m\leq n)\rightarrow(n-m\geq 0\wedge m>0).$$

TCCs are built by a so-called *static analysis* of the specification. Indeed, the premises and conclusions of the above TCCs are built by analyzing the traces given by the conditions and commands in the **then** and **else** branches of the **if then else** instructions:

- On the one side, the conditions in the implications of the TCCs above are built conjugating accordingly either the condition or its negation in the nested **if then else** commands. The first recursive call, $\gcd(m-n,n)$, is executed whenever not $m=n$ and not $m=0$ and $m>n$; the second one, whenever not $m=n$ and not $m=0$ and not $m>n$. Confer this with the premises of the previous TCCs.
- On the other side, the conclusions in the implications of the TCCs above are built from the types of the parameters and the arguments used in the recursive calls. The first recursive call uses as first and second arguments, respectively, $m-n$ and n; thus, it should be guaranteed that the former one is a natural and the second one a positive natural. Similarly, for the second recursive call, it should be guaranteed that $n-m$ and m are, respectively, a natural and a positive natural number.

A second class of TCCs is related with the *termination* or *totality* of the function being specified. For doing this, PVS uses the measure function provided by the user. For the case of gcd, this is the function $(m,n)\mapsto m+n$, as previously mentioned. And the related TCCs should express that this measure strictly decreases for the parameters of the specified function and arguments given in each recursive call:

$$\forall(m:\mathbb{N},n:\mathbb{N}^+):(m\neq n\wedge m\neq 0\wedge m>n)\rightarrow m+n>(m-n)+n$$

$$\forall(m:\mathbb{N},n:\mathbb{N}^+):(m\neq n\wedge m\neq 0\wedge m\leq n)\rightarrow m+n>m+(n-m).$$

We illustrate how the latter termination TCC can be proved using the Gentzen Sequent Calculus.

One starts with the sequent

$$\Rightarrow \forall(m : \mathbb{N}, n : \mathbb{N}^+) : (m \neq n \wedge m \neq 0 \wedge m \leq n) \rightarrow m + n > m + (n - m).$$

Applying twice the rule ($R_\forall$), one obtains the sequent

$$\Rightarrow (m' \neq n' \wedge m' \neq 0 \wedge m' \leq n') \rightarrow m' + n' > m' + (n' - m'),$$

where m' and n' are *Skolem constants*, which are new arbitrary variable names obtained after eliminating the universal quantifiers from the succedent. Then, applying the rules ($R_\rightarrow$), (LC), and ($L_\wedge$) twice, one obtains the sequent

$$m' \neq n', \quad m' \neq 0, \quad m' \leq n' \Rightarrow m' + n' > m' + (n' - m').$$

The conclusion in the last sequent simplifies to $m' + n' > n'$, which holds since m' is a natural different from zero.

Exercise 51 Consider the PVS specification for gcd below (cf. algorithm 2, $\gcd_2$, in the introduction). This specification of gcd maintains the greatest non-null parameter as second argument in the recursive calls by switching the parameters.

```
gcd_sw(m : posnat, n : nat) : recursive nat =
if n = 0 then
  | m
else
  | if m > n then
  |   | gcd_sw(n, m)
  | else
  |   | gcd_sw(m, n − m)
  | end
end
measure lex2(m, n)
```

Algorithm 4: Specification of gcd with parameter switching in PVS

In first place, specify and prove the TCCs related with type preservation for this specification.

In second place, specify and prove termination TCCs, related with the well-definedness of the specified function $\gcd_{sw}$. Notice that the measure used now is $(m, n) \mapsto \texttt{lex2}(m, n)$ instead $(m, n) \mapsto m + n$, which was adequate for the previous specification of gcd; indeed the latter one does not work for the first (parameter switching) recursive call ($\neg(m + n > n + m)$). For the selected measure function `lex2`, the ordering is given as the lexicographic ordering on the parameters:

$$\texttt{lex2}(x, y) > \texttt{lex2}(u, v) \textbf{ iff } x > u \vee (x = u \wedge y > v)$$

Now, specify and prove termination TCCs.

Formulas are used to specify or enunciate conjectures, lemmas, corollaries, and theorems. For instance, below we enunciate the conjecture that the specified function $\gcd_{sw}$ commutes for positive naturals, named `gcd_sw_commutes`:

$$\texttt{gcd\_sw\_commutes CONJECTURE } : \forall(m, n : posnat) : \gcd_{sw}(m, n) = \gcd_{sw}(n, m).$$

Once this conjecture is proved, it can be labeled as `LEMMA`, `COROLLARY`, or `THEOREM`. This is the suggested discipline, but not an obligation. The name of each conjecture is selected by the user.

4.1.2 The PVS Proof Commands Versus Gentzen Sequent Rules

A minimal subset of logical proof commands of PVS are presented in this section and their relation with the Gentzen Sequent Calculus of the classical predicate logic is explained. Sequents are written with notation $\Gamma \mid ---\Delta$, and premises, $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$, and conclusions, $\Delta = \{\delta_1, \ldots, \delta_n\}$, now separated by the symbol $\mid ---$, are labeled with different negative and positive naturals between square brackets as below:

$$\begin{array}{l} [-1]\ \gamma_1 \\ \quad \vdots \\ [-m]\ \gamma_m \\ \mid --- \\ [1]\ \delta_1 \\ \quad \vdots \\ [n]\ \delta_n. \end{array}$$

Initially, we illustrate how the first termination TCC of the specification of gcd in Algorithm 3 is proved in PVS, relating Gentzen sequent deductive rules and PVS proof commands.

The PVS prover starts with the sequent

$$\mid ---[1]\ \forall(m : \mathbb{N}, n : \mathbb{N}^+) : (m \neq n \wedge m \neq 0 \wedge m \leq n) \rightarrow m + n > m + (n - m).$$

Applying the command `(skolem! 1)`, which corresponds to applications of the rule $(R_\forall)$, one obtains the sequent

$$\mid ---[1]\ (m' \neq n' \wedge m' \neq 0 \wedge m' \leq n') \rightarrow m' + n' > m' + (n' - m').$$

Then, applying the command `(flatten)`, which in this situation executes the rules ($R_{\rightarrow}$) and (LC) once, and ($L_{\wedge}$) twice, one obtains the sequent

$$
\begin{array}{l}
[-1]\ m' \neq n' \\
[-2]\ m' \neq 0 \\
[-3]\ m' \leq n' \\
\mid \text{---} \\
[1]\ m' + n' > m' + (n' - m').
\end{array}
$$

The conclusion in the last sequent simplifies to $m' + n' > n'$, which holds since m' is a natural different from zero. To finish it is enough to apply the command `(assert)`. The last command applies the algebraic engine of PVS, which consists of an exhaustive collection of properties specified and proved correct in the prelude library of this proof assistant.

Although it appeared in the previous example, it is necessary to stress to the reader that formal proofs start from the target formula. Thus, rules of the Gentzen Sequent Calculus are applied in bottom-up manner. This should be considered in the sequel when proof commands and Gentzen sequent rules are related.

Another important aspect to be considered in the organization of formulas in the PVS sequents is that the rules of *c-equivalence* are automatically applied in such a manner that formulas with the symbol of negation $\neg$ as their heading logical connective, neither will appear in the premises nor in the conclusions, but instead without their heading negation symbols, respectively, in the conclusions or premises of the sequent. In other words, any negated formula either in the premises or in the conclusions is, respectively, moved to the conclusions or to the premises eliminating its heading negation connective.

For illustrating this, consider what will happen when one applies the command `(prop)` to a sequent as below. This command repeatedly applies the logical propositional rules, i.e., ($L_{\wedge}$), ($L_{\vee}$), ($L_{\rightarrow}$), ($R_{\wedge}$), ($R_{\vee}$), and ($R_{\rightarrow}$), and axioms, i.e., (Ax) and ($L_{\perp}$), until no longer possible.

$$
\begin{array}{l}
\vdots \\
[-i]\ A \wedge \neg B \\
\vdots \\
\mid \text{---} \\
\vdots \\
[j]\ (\neg C) \rightarrow D \\
\vdots
\end{array}
\qquad
\begin{array}{c}
\\
\\
\\
\texttt{(prop)} \\
\rightsquigarrow \\
\\
\\
\\
\end{array}
\qquad
\begin{array}{l}
[-1]\ A \\
\vdots \\
\mid \text{---} \\
[1]\ B \\
[2]\ C \\
[3]\ D \\
\vdots
\end{array}
$$

Axioms

Axioms (Ax) and ($L_{\perp}$) are applied automatically always when possible to the current (active) sequent. Usually it is necessary to apply PVS proof commands such

as `(assert)` or `(prop)` in order to detect that the sequent under consideration gives an instance of an axiom and after that, the proof successfully concludes. For instance when one applies the command of propositional simplification `(prop)` to the sequent below, PVS concludes automatically applying the propositional rules and (Ax):

$$\begin{array}{l}\texttt{|---}\\ [1]A \wedge \neg B \wedge \neg C \rightarrow A\end{array} \qquad \begin{array}{c}\texttt{(prop)}\\ \rightsquigarrow\end{array} \qquad \texttt{Q.E.D.}$$

In the previous proof, the intermediate sequent, in the middle below, is generated but immediately it is discharged concluding "`Q.E.D.`" (from Latin *quod erat demonstrandum*, i.e., "what was to be demonstrated"), since it is an instance of the Gentzen sequent rule (Ax).

$$\begin{array}{l}\texttt{|---}\\ [1]A \wedge \neg B \wedge \neg C \rightarrow A\end{array} \quad \begin{array}{c}\texttt{(prop)}\cdots\\ \rightsquigarrow\end{array} \quad \begin{array}{l}[-1]\ A\\ \vdots\\ \texttt{|---}\\ [1]\ B\\ [2]\ C\\ [3]\ A\\ \vdots\end{array} \quad \begin{array}{c}\cdots\texttt{(prop)}\\ \rightsquigarrow\end{array} \quad \texttt{Q.E.D.}$$

In general, after applying proof commands, axioms are applied whenever possible to the active sequent.

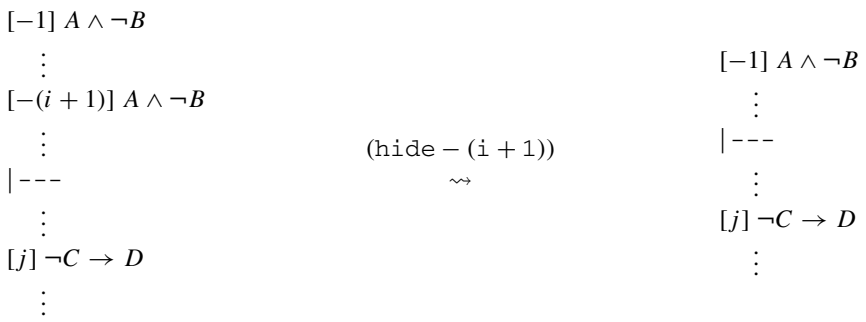
Structural Commands

The application of structural rules (LW) and (RW) (for weakening), and (LC) and (RC) (for contraction) is done, respectively, by PVS proof commands `(copy)` and `(hide)`.

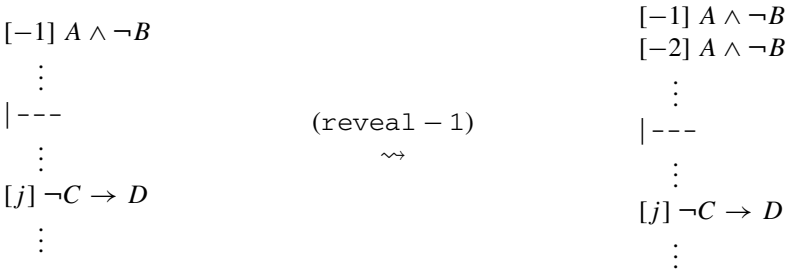
The proof command `(copy)` duplicates a formula selected either from the antecedent or the succedent of the current sequent. For instance,

$$\begin{array}{l}\vdots\\ [-i]\ A \wedge \neg B\\ \vdots\\ \texttt{|---}\\ \vdots\\ [j]\ \neg C \rightarrow D\\ \vdots\end{array} \qquad \begin{array}{c}\texttt{(copy - i)}\\ \rightsquigarrow\end{array} \qquad \begin{array}{l}[-1]\ A \wedge \neg B\\ \vdots\\ [-(i+1)]\ A \wedge \neg B\\ \vdots\\ \texttt{|---}\\ \vdots\\ [j]\ \neg C \rightarrow D\\ \vdots\end{array}$$

The proof command `(hide)` hides a formula selected either from the antecedent or the succedent of the current sequent. For instance,



It is important to stress that PVS does not get rid of hidden formulas. Indeed, the command `(hide)` is more general than the weakening rules because one can hide any formula, even it does not appear duplicated in the context. For this reason, hidden formulas remain invisible but always available, and can be recovered by application of the meta-command `show-hidden-formulas`, which is used to visualize all hidden formulas (and their indices) and the proof command `(reveal)`, which is applied using as parameter the index of the selected hidden formula. For instance, applying the meta-command `show-hidden-formulas` after the last example, one checks that the sole hidden formula is the formula in the antecedent indexed now as $[-1]\ A \wedge \neg B$. Then this formula can be recovered as illustrated below. Notice how indexation changes.



Logical Commands

The proof command `(flatten)` repeatedly applies rules ($L_\wedge$), ($R_\vee$) in order to separate formulas in a conjunction in the antecedent as well as in a disjunction in the succedent of a sequent. Also, `(flatten)` repeatedly applies the rule ($R_\rightarrow$) in order to move premises and conclusions of an implicational formula in the succedent to the antecedent and succedent, respectively. This is done exhaustively and always trying to apply axioms automatically. Check the example below in which the PVS proof command `(flatten)` applies twice the rule ($R_\rightarrow$) and once each of the rules ($L_\wedge$) and ($R_\vee$).

$$
\begin{array}{l}
\vdash\!\!\text{---} \\
[1]\ A \wedge B \to (C \vee D \to C \vee (A \wedge C))
\end{array}
\qquad
\begin{array}{c}
\texttt{(flatten)} \\
\rightsquigarrow
\end{array}
\qquad
\begin{array}{l}
[-1]\ A \\
[-2]\ B \\
[-3]\ C \vee D \\
\vdash\!\!\text{---} \\
[1]\ C \\
[2]\ A \wedge C
\end{array}
$$

Since PVS works modulo c-equivalence and exhaustively applies (Ax), applying `(flatten)` to the objective formula $A \wedge B \to (C \vee D \to C \vee \neg(A \wedge C))$ will conclude with `Q.E.D.` automatically.

Exercise 52 What is the result of applying the PVS proof command `(flatten)` to the objective formulas below?

1. $(A \wedge B \to C \vee D) \to C \vee \neg(A \wedge C))$.
2. $(A \wedge B \to C \vee D) \to C \vee (A \wedge C))$.

In contrast with the command `(flatten)`, the PVS proof command `(split)` is used to repeatedly apply branching proof rules as ($R_\wedge$), ($L_\vee$) and ($L_\to$), which are Gentzen sequent rules that require two premises. This command splits the selected formula of the sequent into two or more sequents accordingly to these branching rules. As before, `(split)` applies axioms when possible. Below, the action of this command is exemplified.

$$
\begin{array}{l}
[-1]\ (A \to B) \to A \\
\vdash\!\!\text{---} \\
[1]\ A
\end{array}
\qquad
\begin{array}{c}
\texttt{(split}\ -1\texttt{)} \\
\rightsquigarrow
\end{array}
\qquad
\begin{array}{l}
[-1]\ A \\
\vdash\!\!\text{---} \\
[1]\ A \\
\\
\vdash\!\!\text{---} \\
[1]\ A \to B \\
[2]\ A
\end{array}
$$

Above, the first derivated sequent $A \vdash\!\!\text{---} A$ is included for a matter of clarity. Indeed, `(split)` applies automatically (Ax) closing this branch of the derivation.

Exercise 53 Using only the PVS proof commands `(flatten)` and `(split)` prove Peirce's law. That is, prove the sequent $\vdash\!\!\text{---}((A \to B) \to A) \to A$.

Indeed, only one application of the command `(prop)` will close the proof of this sequent since it applies repeatedly logical propositional rules and axioms.

The PVS proof commands `(inst)` and `(skolem)` are related with Gentzen right and left sequent rules for quantifiers. The proof command `(skolem)` applies Gentzen sequent rules ($R_\forall$) and ($L_\exists$) replacing variables under universal quantification of formulas in the succedent and existential quantification in the antecedent of the current sequent, respectively, by Skolem constants or *fresh* variables, i.e., by new arbitrary variables that do not appear in the derivation in progress. For instance, consider a unary predicate P over a type domain T. After applying `(flatten)`

to the sequent $|---\forall_{x:T} : P(x) \to \neg\exists_{x:T} : \neg P(x)$ one obtains the sequent to the left below, which can be proved by adequate applications of the PVS commands `(skolem)` and `(inst)` as shown below:

$$\begin{array}{l}[-1]\ \forall_{x:T} : P(x)\\ [-2]\ \exists_{x:T} : \neg P(x)\\ |---\end{array} \quad \begin{array}{c}\texttt{(skolem -2 "z")}\\ \rightsquigarrow\end{array} \quad \begin{array}{l}[-1]\ \forall_{x:T} : P(x)\\ |---\\ [1]\ P(z)\end{array} \quad \begin{array}{c}\texttt{(inst -1 "z")}\\ \rightsquigarrow\end{array} \quad \text{Q.E.D.}$$

Exercise 54 Which PVS commands are necessary to prove the following sequents:

1. $|---\forall_{x:T} : P(x) \leftrightarrow \neg\exists_{x:T} : \neg P(x)$;
2. $|---\exists_{x:T} : P(x) \leftrightarrow \neg\forall_{x:T} : \neg P(x)$.

In the previous proofs, the order in which instantiation and skolemization are applied is crucial. In principle skolemization should be applied first, so that the Skolem constants can be used as parameters of the instantiation; otherwise a specific term should be available to be provided as parameter of the instantiation. To illustrate better this, consider the sequent $|---\forall_{x:T} : P(x) \to \exists_{x:T} : P(x)$. After application of the PVS command `(flatten)` one obtains the sequent below, for which only instantiation (correspondingly, rules ($L_\forall$) and ($R_\exists$)) is possible:

$$\begin{array}{l}[-1]\ \forall_{x:T} : P(x)\\ |---\\ [1]\ \exists_{x:T} : P(x).\end{array}$$

Observe that the type T requires to be a nonempty type in order to validate this sequent. In the case T is an empty type the antecedent will be true, but the succedent false. This is a particular issue in computer science that is avoided in the theory because one assumes that the domain of interpretation should be a nonempty set. Supposing T is a nonempty type and one has a declared constant of type T, say $c : T$, the proof proceeds as below:

$$\begin{array}{l}[-1]\ \forall_{x:T} : P(x)\\ |---\\ [1]\ \exists_{x:T} : P(x)\end{array} \quad \begin{array}{c}(inst\ -1\ "c")\\ \rightsquigarrow\end{array} \quad \begin{array}{l}[-1]\ P(c)\\ |---\\ [1]\ \exists_{x:T} : P(x)\end{array} \quad \begin{array}{c}\texttt{(inst 1 "c")}\\ \rightsquigarrow\end{array} \quad \text{Q.E.D.}$$

Skolemization and instantiation could be performed in a more automatic manner by applications of the commands `(skeep)` and `(inst?)`. The former essentially is the same as `(skolem)`, but maintaining the name of the quantified variables, whenever these names are fresh variable names in the current sequent, and the latter is essentially `(inst)`, but trying to instantiate quantified variables with terms that appear in the context of the current sequent. The former command also applies propositional transformations such as those involved in the commands `(flatten)` and `(split)`.

The (Cut) Rule and Associated PVS Proof Commands

The (Cut) rule is applied in several situations through commands such as `(case)`, `(lemma)`, and `(rewrite)`.

The proof command `(case)` introduces a new premise in the current sequent, but always splitting the current proof into two branches: the first one assuming the condition given as argument of the command `(case)`, and the second one being its negation. To illustrate its use, suppose one wants to prove that the operator $\gcd_{sw}$ specified in Algorithm 4 is commutative for positive naturals as enunciated in the conjecture at the end of the previous section.

`gcd_sw_commutes : CONJECTURE` $\forall(m, n : posnat) : \gcd_{sw}(m, n) = \gcd_{sw}(n, m).$

The proof can be divided into two cases according to whether the first argument is greater than or equal to the second one or not. Thus, after application of the command `(skolem)` one obtains the sequent below that branches into two sequents by application of `(case)`:

$$\begin{array}{l}|\text{---}\\ [1]\ \gcd_{sw}(m, n) = \gcd_{sw}(n, m)\end{array} \quad \begin{array}{c}(\texttt{case "m} \geq \texttt{n"})\\ \rightsquigarrow\end{array} \quad \begin{array}{l}[-1]\ m \geq n\\ |\text{---}\\ [1]\ \gcd_{sw}(m, n) = \gcd_{sw}(n, m)\\ \\ |\text{---}\\ [1]\ m \geq n\\ [2]\ \gcd_{sw}(m, n) = \gcd_{sw}(n, m).\end{array}$$

From the perspective of (Cut) in the Gentzen Sequent Calculus, notice that what one has is exactly the following application of this rule:

$$\frac{\Rightarrow \gcd_{sw}(m, n) = \gcd_{sw}(n, m), m \geq n \qquad m \geq n \Rightarrow \gcd_{sw}(m, n) = \gcd_{sw}(n, m).}{\Rightarrow \gcd_{sw}(m, n) = \gcd_{sw}(n, m)}\ (\text{Cut})$$

The PVS proof commands `(lemma)` and `(rewrite)`, respectively, invoke and try to apply a lemma previously stated. For instance suppose one has the following formalized result, lemma `l1`:

`l1 : LEMMA` $\forall(i, j : posnat) : i \geq j \rightarrow \gcd_{sw}(i, j) = \gcd_{sw}(j, i).$

Then, to use this lemma in order to prove the previous sequent one applies the command `(lemma)` as below, where $\varphi(i, j)$ denotes $i \geq j \rightarrow \gcd_{sw}(i, j) = \gcd_{sw}(j, i)$:

$$\begin{array}{l}|\text{---}\\ [1]\ \gcd_{sw}(m, n) = \gcd_{sw}(n, m)\end{array} \quad \begin{array}{c}(\texttt{lemma"l1"})\\ \rightsquigarrow\end{array} \quad \begin{array}{l}[-1]\ \forall(i, j : posnat) : \varphi(i, j)\\ |\text{---}\\ [1]\ \gcd_{sw}(m, n) = \gcd_{sw}(n, m).\end{array}$$

From this point, an adequate application of the command `(inst)` followed of `(split)` to the formula in the antecedent (indeed, `(inst -1 "m" "n")`) will give rise to two objectives: one of them is trivial, which is automatically proved by simple application of (Ax), and the other, related with the case of the proof in which $\neg(m \geq n)$ holds.

$$\begin{array}{ccccc} & & & & [-1]\ \gcd_{sw}(m,n) = \gcd_{sw}(n,m) \\ & & & & |\ --- \\ (\texttt{inst}-1\texttt{"m""n"}) & & (\texttt{split}) & & [1]\ \gcd_{sw}(m,n) = \gcd_{sw}(n,m) \\ \rightsquigarrow & \cdots & \rightsquigarrow & & \\ & & & & |\ --- \\ & & & & [1]\ m \geq n \\ & & & & [2]\ \gcd_{sw}(m,n) = \gcd_{sw}(n,m). \end{array}$$

From the theoretical point of view, what one does when a lemma is invoked is applying the rule (Cut) using the proof of the invoked lemma. Suppose ∇_{l1} is the proof of lemma `l1`, then one has the proof sketch below. Since the proof ∇_{l1} is ready (or assumed), to prove the target sequent, that is $\Rightarrow \gcd_{sw}(m,n) = \gcd_{sw}(n,m)$, what is necessary is only to prove the premise sequent to the right, that is $\forall(i,j)$: $\varphi(i,j) \Rightarrow \gcd_{sw}(m,n) = \gcd_{sw}(n,m)$.

$$\dfrac{\overset{\nabla_{l1}}{\Rightarrow \forall(i,j) : \varphi(i,j)} \qquad \forall(i,j) : \varphi(i,j) \Rightarrow \gcd_{sw}(m,n) = \gcd_{sw}(n,m).}{\Rightarrow \gcd_{sw}(m,n) = \gcd_{sw}(n,m)}\ (\text{Cut})$$

The PVS proof rule `(rewrite)` similar to `(lemma)` invokes a lemma, but it additionally tries to instantiate it adequately in an automatic manner. Actually, applications of `(rewrite)` do not necessarily would instantiate as expected by the user. Thus, it would require special attention.

Table 4.1 summarizes the relations between Gentzen sequent rules and PVS proof commands. The table also includes the correspondences between Gentzen Sequent and Natural Deduction rules as given in the previous chapter. Indeed, it is important to remember here that according to the proof of equivalence between deduction *à la* Gentzen Sequent Calculus and Deduction Natural, elimination (natural deduction) rules relate with the corresponding left Gentzen sequent rules using (Cut) (revise Theorem 13). Marks in the second column indicate that rules (Ax) and ($L_\perp$) as well as c-equivalence derivations are automatically applied whenever possible when the proof commands are applied.

Table 4.1 SC and ND rules versus PVS proof commands

	(Ax)($L_\perp$) c-equiv	(LW) (RW)	(LC) (RC)	($L_\wedge$) ($\wedge_e$)	($L_\vee$) ($\vee_e$)	($L_\to$) ($\to_e$)	($L_\forall$) ($\forall_e$)	($L_\exists$) ($\exists_e$)	($R_\wedge$) ($\wedge_i$)	($R_\vee$) ($\vee_i$)	($R_\to$) ($\to_i$)	($R_\forall$)	($R_\exists$) ($\forall_i$)	(Cut) ($\exists_i$)
(hide)		×												
(copy)			×											
(flatten)	✓			×						×	×			
(split)	✓				×	×			×					
(skolem)	✓							×				×		
(inst)	✓						×						×	
(lemma)														×
(case)	✓													×

4.2 PVS Proof Commands for Equational Manipulation

Formalization of properties of specified operators and functions requires equational manipulation. Here, we will explain how defined operators are expanded and their definitions lifted inside logical formulas. A few PVS proof commands will be commented: `(expand)`, `(lift-if)`, `(replace)`, `(replaces)`, `(grind)`, `(decompose-equality)`, and `(apply-extensionality)`.

As an elementary example, consider the definition of the absolute value of real numbers, specified in PVS as

$$abs(x : \text{real}) : \text{nonneg\_real} = \textbf{if}\ \ x < 0\ \ \textbf{then}\ \ -x\ \ \textbf{else}\ \ x\ \ \textbf{endif}.$$

And suppose we have the target of proving the triangle inequality:

$$|\text{---}\ \ \textbf{[1]}\ \ \ \textbf{forall}(x, y : \text{real}) : abs(x) + abs(y) \geq abs(x + y).$$

To prove this, it is required to use the definition of the operator abs, which is done through *expansions* of the definition of this operator according to its specification. Notice that we are applying an equational conversion that is neither included in the Natural Deduction nor in the Gentzen Sequent Calculus.

After two applications of `(skolem)`, one can apply the command `(expand "abs" 1 1)` which substitutes the name `abs` in the formula 1, and specifically the first occurrence from left to right in that formula, by its definition:

$$\underset{\rightsquigarrow}{\texttt{(skolem"x""y")}} \quad \ldots \quad \underset{\rightsquigarrow}{\texttt{(expand"abs"1 1)}} \quad \begin{array}{l}|\text{---}\\ [1]\ \textbf{if}\ \ x < 0\ \ \textbf{then}\ \ -x\ \ \textbf{else}\ \ x\ \ \textbf{endif} + \\ \qquad\qquad abs(y) \geq abs(x + y).\end{array}$$

More generally, one can expand several occurrences, say $i_1, \ldots, i_m$, of an operator, say `f`, in the formula n, applying the command `(expand "f" n (`$i_1 \ldots i_m$`))`.

This kind of equational transformation is related with natural deduction inference rules such as $(=_i)$ and $(=_e)$ and Gentzen inference rules such as $(R_=)$ and $(L_=)$ given below:

$$\frac{s = t \quad \varphi[x/s]}{\varphi[x/t]}\ (=_e) \qquad \frac{}{t = t}\ (=_i)$$

$$\frac{\Gamma \Rightarrow \Delta, \varphi[x/s]}{s = t, \Gamma \Rightarrow \Delta, \varphi[x/t]}\ (\mathrm{L}_=) \qquad \frac{}{\Gamma \Rightarrow \Delta, t = t}\ (\mathrm{R}_=).$$

From these rules it is possible to prove that $=$ is symmetric and transitive. For instance, symmetry of $=$ can be derived as below, for the Gentzen sequent rules:

$$\frac{\Rightarrow (x = s)[x/s]\ \ (\mathrm{R}_=).}{s = t \Rightarrow t = s}\ (\mathrm{L}_=)$$

Thus, we will not distinguish from left- and right-hand side of equations.

Exercise 55

a. Prove that the relation $=$ is transitive for the Gentzen sequent rules.
b. Prove that the relation $=$ is symmetric and transitive for the rules in deduction natural.

Notice that in this case, the rule $(\mathrm{R}_=)$ is applied with the equation given by the definition of abs: $abs(x) =$ **if** $x < 0$ **then** $-x$ **else** x **endif**, and to get rid of this equation in the conclusion, a (Cut) is applied using also as premise the sequent associated with this definition, that is $|\text{---}\ abs(x) =$ **if** $x < 0$ **then** $-x$ **else** x **endif**.

The next step to proceed with the proof of the triangle inequality is lifting in the definition of $abs(x)$ in the succedent formula, which is done through application of the PVS command `(lift-if)` obtaining the sequent below:

$$\ldots \quad \underset{\rightsquigarrow}{(\texttt{lift-if})} \quad \begin{array}{l} |\text{---} \\ [1]\ \textbf{if}\ \ x < 0\ \ \textbf{then}\ \ -x + abs(y) \geq abs(x + y) \\ \qquad \textbf{else}\ \ x + abs(y) \geq abs(x + y) \\ \qquad \textbf{endif}. \end{array}$$

Finally, by application of the command `(split)` one obtains to subobjective sequents:

$$\ldots \quad \underset{\rightsquigarrow}{(\texttt{split})} \quad \begin{array}{l} |\text{---} \\ [1]\ x < 0\ \ \textbf{implies}\ \ -x + abs(y) \geq abs(x + y) \\ \\ |\text{---} \\ [1]\ \ \textbf{not}\ \ x < 0\ \ \textbf{implies}\ \ x + abs(y) \geq abs(x + y). \end{array}$$

From the rule ($L_{=}$) and by applying twice c-equivalences, one has a derived rule that replaces the terms in an equality occurring in any formula in the antecedent:

$$\dfrac{\dfrac{\dfrac{\Gamma, \varphi[x/s] \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg\varphi[x/s]}\ \text{(c-equivalence)}}{s \doteq t, \Gamma \Rightarrow \Delta, \neg\varphi[x/t]}\ (L_{=})}{s \doteq t, \Gamma, \varphi[x/t] \Rightarrow \Delta}\ \text{(c-equivalence)}$$

Equational replacement is performed in PVS by applying the command `(replace i j)`, where i is the label of the equality and j is the label of the formula (either in the antecedent or in the succedent) of the sequent in which the replacement is applied. As default, the replacement of the left-hand side by the right-hand side of the equality is applied, but this can be modified including the parameter "`rl`": `(replace i j rl)`.

The command `(replaces)` is a variant of `(replace)` that iterates the replacement given by the selected equality and afterward hides the equality.

The last command to be discussed in this section is `(grind)`. This command will apply all possible equational definitional expansions and lift in all definitions over the logical formulas. In addition, `(grind)` applies all logical rules associated with `(prop)` and `(assert)`. Indeed the triangle inequality might be proved by a simple application of `(grind)`.

Exercise 56 Verify in PVS that the triangle inequality is concluded by a simple application of `(grind)`. Also, complete the formalization of the triangle inequality by application of the PVS command `(expand)` to the three occurrences of *abs* in the target objective and repeatedly application of the command `(lift-if)` to lift in these definitions inside the target formula. To conclude you also would need to apply either the command `(split)` or `(prop)` in order to split the target objective into simpler subobjectives. Applications of `(assert)` will also be required to deal with the algebra of inequalities over reals.

The last two PVS commands to be treated here, related with equational manipulation, are `(decompose-equality)` and `(apply-extensionality)`. These commands deal with equality between expressions according to the structure of their types. The former command decomposes equality according to the abstract data structure of the terms involved in an equality. As a first example, consider the data structure of finite sequences given in the PVS prelude as below, where `T` is a non-interpreted type:

```
finite_sequence : TYPE = [# length: nat, seq: [below[length] ->
T] #].
```

Thus, a finite sequence consists of two parts; the former is the `length` of the sequence and the latter `seq`, that is, the sequence itself, is a function from indices below the length of the sequence (i.e., from 0 to the length minus one), to objects of type `T`.

Consider now terms s and t of type `finite_sequence[bool]` and the sequent $|\text{---}[1]\, s = t$. Applying the command `(decompose-equality 1)` this

equation will be decomposed into two equations for the length and the sequences itself:

```
|---                                          |---
[1] s = t      (decompose-equality 1)         [1] s`length = t`length]
                      ⇝
                                              |---
                                              [1] s`seq = t`seq
```

The last sequent can be further decomposed as below:

```
|---                  (decompose-equality 1)      |---
[1] s` seq = t` seq              ⇝                 [1] s` seq(i) = t` seq(i),
```

where the variable `i` is a natural below the length of s: `i` $\leq$ s`` `length``. This information about the type of any term can be recovered in PVS with the command `(typepred i)`.

For another example on decomposition of equalities, consider a datatype for lambda terms with variables, applications, and abstractions (respectively, `vars`, `appl`, and `abs`) specified as below:

```
term[variable : TYPE+]: DATATYPE
BEGIN
 vars (v: variable) : var?
 abs  (v: variable, body: term): abs?
 app  (tf: term, ta: term): app?
END term.
```

Abstractions have two elements: `v` the variable of the abstraction and the `body` of the abstraction that is also a term. Applications have two elements: `tf` that is the functional part of the application and `ta` that is the argument of the application; both `tf` and `ta` are terms.

Now consider one has an equality between lambda terms u and v that are applications:

$$\texttt{app?}(u) \ , \ \texttt{app?}(v) \,|\texttt{---}u = v.$$

In this case by application of extensionality with the command `(apply-extensionality)` the proof is split into two subobjectives on the equality of the functional and argument components of the applications u and v:

```
[–1] app?(u)
[–2] app?(v)
|---
[1] u = v
```

(apply-extensionality 1)

⇝

```
[–1] app?(u)
[–2] app?(v)
|---
[1] tf(u) = tf(v)

[–1] app?(u)
[–2] app?(v)
|---
[1] ta(u) = ta(v).
```

Exercise 57 Explore the application of the last two studied commands `(decompose-equality)` and `(apply-extensionality)` to equalities between objects of type `list`. As usual, in PVS, a list l over the non-interpreted type `T`, that is, `list[T]`, might be either empty or a cons, checked as `null?(`l`)` and `cons?(`l`)`, respectively. Thus, data structures of lists own objects that are empty lists `null` or built recursively using any element of type `T`, say a, and a list, say l, as `cons(`a, l`)`. The head of a nonempty list is computed as `car(`l`)` and the tail as `cdr(`l`)`. How will you deal with an equation of the form $l1 = l2$ between nonempty lists?

4.3 Proof Commands for Induction

Several commands are available for induction from which we will explain two: `(induct)` and `(measure-induct+)`. These commands build an induction scheme according to the data structure associated to the variable given as argument. The former builds an induction scheme based on the type of the variable given as parameter and provides the base and induction cases. The latter builds an induction scheme using the strong or complete induction principle instantiated with a measure provided by the user. The induction scheme follows the data structure of the variables on which the measure is defined.

To illustrate the use of `(induct)`, consider the following specification of the sum of the f-images of the first n naturals: $\sum_{i=0}^{n} f(n)$.

```
sum(f : [nat -> nat], n : nat) : recursive nat =
if n = 0 then
| f(0)
else
| f(n) + sum(f, n − 1)
end
 measure n
```

Algorithm 5: Specification of sum of the n first f-images in PVS

Using *sum* it is possible to compute the sums in Sect. 1.3: $\sum_{i=1}^{n} i$ as $sum(x \mapsto x, n)$ and $\sum_{i=0}^{n} k^i$ as $sum(x \mapsto k^x, n)$. Indeed, the functions $x \mapsto x$ and $x \mapsto k^x$ are specified as lambda functions of the respective forms:

$$(lambda(x : \text{nat}) : x) \text{ and } (lambda(x : \text{nat}) : k^x).$$

To illustrate the use of the command `(induct)`, let us consider the proof of the following equality:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}.$$

Using *sum*, the sequent associated with this equality is given as below:

$$|\text{---} \quad \textbf{forall} \ (n : \text{nat}) : sum((lambda(x : \text{nat}) : x \hat{} 2), n) = \frac{n(n+1)(2n+1)}{6}.$$

After applying the command `(induct"n")`, PVS builds the induction scheme for the natural variable n. Thus, two subgoals are obtained, the first one associated with the induction basis and the second one with the inductive step:

$$|\text{---} [1] \ sum((lambda(m : \text{nat}) : m \hat{} 2), 0) = \frac{0(0+1)(2\ 0+1)}{6}$$

and

$$\begin{aligned} |\text{---} [1] \ & \textbf{forall} \ j : \\ & sum((lambda(m : nat) : m \hat{} 2), j) = \frac{j(j+1)(2j+1)}{6} \ \textbf{implies} \\ & sum((lambda(m : \text{nat}) : m \hat{} 2), j+1) = \frac{(j+1)(j+1+1)(2(j+1)+1)}{6}. \end{aligned}$$

The subobjective associated with the induction basis is proved by applying the command `(assert)` and the command `(expand)` to expand the definitions of *sum*, power ($\hat{}$), and *expt*. The last operator is used in the prelude of PVS to define power. Indeed, application of the command `(grind)` will be enough to solve this equality.

The subobjective related with the inductive step derives the objective below applying the commands `(skolem)` and `(flatten)`:

$$\begin{aligned} & [-1] \ sum((lambda(m : nat) : m \hat{} 2), j) = \frac{j(j+1)(2j+1)}{6} \\ |\text{---} \\ & [1] \ sum((lambda(m : \text{nat}) : m \hat{} 2), j+1) = \frac{(j+1)(j+1+1)(2(j+1)+1)}{6}. \end{aligned}$$

After expanding the definition of *sum* in the succedent by application of the command `(expand "sum" 1)` one obtains the following:

$$
\begin{array}{l}
\quad [-1]\ sum((lambda(m : nat) : m \hat{}\ 2), j) = \dfrac{j(j+1)(2j+1)}{6} \\
|\text{---} \\
\quad [1]\ sum((lambda(m : \text{nat}) : m \hat{}\ 2), j) + (j+1) \hat{}\ 2 = \dfrac{(j+1)(j+1+1)(2(j+1)+1)}{6}.
\end{array}
$$

Then, by replacing the left-hand side of the equation in the antecedent by the right-hand side in the succedent, applying the PVS command `(replace -1 1)` one obtains the sequent

$$
\begin{array}{l}
\quad [-1]\ sum((lambda(m : nat) : m \hat{}\ 2), j) = \dfrac{j(j+1)(2j+1)}{6} \\
|\text{---} \\
\quad [1]\ \dfrac{j(j+1)(2j+1)}{6} + (j+1) \hat{}\ 2 = \dfrac{(j+1)(j+1+1)(2(j+1)+1)}{6}.
\end{array}
$$

As for the basis, this subobjective is proved by applying the commands `(assert)` and `(expand)`. The latter used to expand the definitions of *sum*, power ($\hat{}$), and *expt*. Similarly, the application of the command `(grind)` will be enough to solve this query.

For illustrating a more elaborated inductive scheme built by the command `(induct)` consider the sequent below, where `term` is the data structure given in the previous section for lambda terms, and suppose we have a predicate $p?$ on lambda terms:

$$|\text{---}[1]\ \texttt{forall}\ (t : \text{term}) : p?(t).$$

The application of the command `(induct "t")` will build an inductive scheme based on the structure of lambda terms, considering variables, abstractions, and applications, according to the datatype. Thus, three subobjectives are built:

`(induct "t")` $\rightsquigarrow$

$$
\left\{
\begin{array}{l}
|\text{---} \\
[1]\ \textbf{forall}\ (x : \text{variable}) : p?(vars(x)) \\
\\
|\text{---} \\
[1]\ \textbf{forall}\ (x : \text{variable}, t1 : \text{term[variable]}) : \\
\qquad p?(t1)\ \textbf{implies}\ p?(abs(x, t1)) \\
\\
|\text{---} \\
[1]\ \textbf{forall}\ (t1, t2 : \text{term[variable]}) : \\
\qquad p?(t1)\ \textbf{and}\ p?(t2)\ \textbf{implies}\ p?(app(t1, t2))
\end{array}
\right.
$$

Exercise 58 Specify the sums of the examples given in Sect. 1.3 and prove the equations using the command `(induct)`:

a. $\displaystyle\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$,

b. $\sum_{i=0}^{n} k^i = \frac{k^{n+1}-1}{k-1}$, for $k \neq 1$.

Now, we will study the second command for induction: `(measure-induct+)`. Consider the following specification of the Fibonacci function.

```
fibonacci(n : nat) : recursive nat =
if n ≤ 0 then
 | n
else
 | fibonacci(n − 1) + fibonacci(n − 2)
end
 measure n
```

Algorithm 6: Specification of the Fibonacci function in PVS

Now, we will prove the conjecture below:

$$\texttt{|---[1]}\ \ \textbf{forall}\ \ (n : \text{nat}) : n \geq 8 \quad \textbf{implies} \quad \mathit{fibonacci}(n) \geq 2n.$$

The induction scheme built by `(induct)` cannot be used in a straightforward manner since the induction hypothesis should be applied not only for $n-1$, but also for $n-2$. The command `(measure-induct+ "n" ("n"))` will build the required complete induction scheme. The first parameter of this command, "n," is a measure function to be used in the induction scheme and, the second parameter, `("n")`, is the list of variables used to build this measure. Applying this command to the sequent one has the following derivation:

`(measure-induct+ "n" ("n"))` ⇝

[−1] **forall** (m : nat) : $m < n$ **implies** $\mathit{fibonacci}(m) \geq 2m$
[−2] $n \geq 8$
|---
[1] $\mathit{fibonacci}(n) \geq 2n$.

Copying the formula [–2] and instantiating the hypotheses with $n-1$ and $n-2$, one obtains the premises required to complete the proof:

`(copy -1)` ⇝ `(inst -1 "n − 1")` ⇝ `(inst -2 "n − 2")` ⇝

[−1] $n - 1 < n$ **implies** $\mathit{fibonacci}(n-1) \geq 2(n-1)$
[−2] $n - 2 < n$ **implies** $\mathit{fibonacci}(n-2) \geq 2(n-2)$
[−3] $n \geq 8$
|---
[1] $\mathit{fibonacci}(n) \geq 2n$.

Notice that in the last sequent it is implicitly required that both $n-1$ and $n-2$ be less than or equal to 8. Thus, the case in which $n \geq 10$ allows application of the required hypotheses, while the cases $n = 8$ and $n = 9$ should be treated apart by expansion of the definition of the function *fibonacci*.

Exercise 59 Complete the proof of this conjecture.

Now we will illustrate the use of the PVS command `(measure-induct+)` proving that the function $\gcd_{sw}$ specified in Exercise 51, Algorithm 4 satisfies the sequent

$$\vert\text{---}[1]\ \ \textbf{forall}\ (m:\ \text{posnat}, n:\ \text{nat}):\ \text{divides}\,(\gcd_{sw}(m,n),m).$$

The measure function required is `lex2`, that is, the same **measure** function used in the specification of the function $\gcd_{sw}$, with parameters "m" and "n." Thus, the induction should be applied with these parameters: `(measure-induct+ "lex2(m,n)" ("m" "n"))`. `(measure-induct+)` builds a complete induction scheme using this measure giving rise to the sequent below:

`(measure-induct+"lex2(m,n)" ("m" "n"))`

$\rightsquigarrow$

[–1] **forall**(i : posnat, j : nat) :
`lex2`$(i,j) < lex2(m,n)$ **implies**
divides $(\gcd_{sw}(i,j),i)$
$\vert$---
[1] divides $(\gcd_{sw}(m,n),m)$

After expanding the definition of $\gcd_{sw}$ and lifting the branching instruction **if then else** using the PVS command `(lift-if)` one obtains the sequent below:

[–1] **forall**(i : posnat, j : nat) : `lex2`$(i,j) < lex2(m,n)$ **implies** divides $(\gcd_{sw}(i,j),i)$
$\vert$---
[1] **if** $n = 0$ **then** divides (m,m)
else if $m > n$ **then** divides $(\gcd_{sw}(n,m),m)$
else divides $(\gcd_{sw}(m,n-m),m)$
endif
endif

After propositional derivations two interesting cases arise:

[–1] $m > n$
[–2] **forall**(i : posnat, j : nat) : `lex2`$(i,j) < lex2(m,n)$ **implies** divides $(\gcd_{sw}(i,j),i)$
$\vert$---
[1] divides $(\gcd_{sw}(n,m),m)$
[2] $n = 0$

and

[–1] **forall**(i : posnat, j : nat) : `lex2`$(i, j) < lex2(m, n)$ **implies** divides $(\gcd_{sw}(i, j), i)$
|---
[1] [–1] $m > n$
[2] divides $(\gcd_{sw}(n, m), m)$
[3] $n = 0$.

For the former subobjective expanding $\gcd_{sw}$ in [1] one obtains the sequent below:

[–1] $m > n$
[–2] **forall**(i : posnat, j : nat) : `lex2`$(i, j) < lex2(m, n)$ **implies** divides $(\gcd_{sw}(i, j), i)$
|---
[1] divides $(\gcd_{sw}(n, m - n), m)$
[2] $n = 0$.

Then copying the induction hypothesis and instantiating it first as `(inst -1 "`n`" "`m`")` and second as `(inst -3 "`$m - n$`" "`n`")` one obtains, after propositional derivations, the objective below:

[–1] divides $(\gcd_{sw}(m - n, n), m - n)$
[–2] divides $(\gcd_{sw}(n, m), n)$
[–3] $m > n$
|---
[1] divides $(\gcd_{sw}(n, m - n), m)$
[2] $n = 0$.

Since $\gcd_{sw}$ commutes (see lemma `gcd_sw_commutes` in Sect. 4.1.2) for positive naturals, the formula [–1] can be rewritten into the formula [–1] divides $(\gcd_{sw}(n, m - n), m - n)$. Also, by expanding $\gcd_{sw}$ in the formula [–2] one obtains the formula [–2] divides $(\gcd_{sw}(n, m - n), n)$. Then, by expanding divides in the whole sequent one obtains the sequent below:

[–1] **exists** $i : m - n = \gcd_{sw}(n, m - n) * i$
[–2] **exists** $j : n = \gcd_{sw}(n, m - n) * j$
[–3] $m > n$
|---
[1] **exists** $k : m = \gcd_{sw}(n, m - n) * k$
[2] $n = 0$.

Finally, by skolemization (rule ($L_\exists$)) of formulas [–1] and [–2] and adequate instantiation of formula [1] (rule ($R_\exists$), through application of the PVS command `(inst 1 "`$i + j$`")`), one concludes the proof of the former subobjective.

The latter subobjective is simpler. Indeed, instantiating the induction hypothesis as `(inst -1 "`m`" "`$n - m$`")` one obtains the required formula to easily conclude the proof.

Exercise 60 a. Complete the proof and formalize the commutativity of $\gcd_{sw}$:

$$|---[1] \ \texttt{forall} \ (m : \text{posnat}, n : \text{nat}) : n > 0 \ \texttt{implies} \ \gcd_{sw}(m,n) = \gcd_{sw}(n,m)$$

As explained in Sect. 4.1.2, this proof does not require induction.

b. Complete the proof of the conjecture

$$|---[1] \ \texttt{forall} \ (m : \text{posnat}, n : \text{nat}) : \text{divides}\,(\gcd_{sw}(m,n), m).$$

4.4 The Semantics of the PVS Specification Instructions

As previously mentioned the PVS specification language uses branching instructions such as **if then else** and **cases of** and the binding instruction **let in**. Thus, it is necessary to establish the concrete semantics of these instructions. In particular, the semantics of the PVS conditional instruction **if then else** can be made explicit by illustrating its behavior in proofs where it appears in the succedent and in the antecedent:

$$\frac{\dfrac{a, \Gamma \,|---\, \Delta, b}{\Gamma \,|---\, \Delta, a \to b}\ \texttt{(flatten)} \qquad \dfrac{\Gamma \,|---\, \Delta, a, c}{\Gamma \,|---\, \Delta, \neg a \to c}\ \texttt{(flatten)}}{\Gamma \,|---\, \Delta, \textbf{if } a \textbf{ then } b \textbf{ else } c \textbf{ endif}}\ \texttt{(split)}$$

$$\frac{\dfrac{a, b, \Gamma \,|---\, \Delta}{a \wedge b, \Gamma \,|---\, \Delta}\ \texttt{(flatten)} \qquad \dfrac{c, \Gamma \,|---\, \Delta, a}{\neg a \wedge c, \Gamma \,|---\, \Delta}\ \texttt{(flatten)}}{\textbf{if } a \textbf{ then } b \textbf{ else } c \textbf{ endif}, \Gamma \,|---\, \Delta}\ \texttt{(split)}$$

By applying the proof command `(prop)` to **if then else** formulas in the succedent or antecedent of a target sequent one will obtain automatically two corresponding sequents as above.

The **cases of** instruction is treated in a similar manner, considering different cases as nested **if then else** commands. We consider again the datatype of lambda terms.

The derivations below illustrate the PVS semantics of **cases of** instructions occurring in the antecedent and succedent of a target sequent. Notice the use of `(lift-if)` proof command.

$$\frac{\dfrac{\texttt{app?}(s), \Gamma \,|---\, \Delta, a \quad \texttt{abs?}(s), \Gamma \,|---\, \Delta, \texttt{app?}(s), b \quad \Gamma \,|---\, \Delta, \texttt{abs?}(s), \texttt{app?}(s), c}{\Gamma \,|---\, \Delta, \textbf{if } \texttt{app?}(s) \textbf{ then } a \textbf{ elsif } \texttt{abs?}(s) \textbf{ then } b \textbf{ else } c \textbf{ endif}}\ \texttt{(prop)}}{\Gamma \,|---\, \Delta, \textbf{cases } s \textbf{ of } \texttt{app}(x,t) : a, \texttt{abs}(t,u) : b, \texttt{vars}(t) : c \textbf{ endcases}}\ \texttt{(lift-if)}$$

$$\dfrac{\dfrac{\texttt{app?}(s), a, \Gamma \mid \texttt{---}\Delta \quad \texttt{abs?}(s), b, \Gamma \mid \texttt{---}\Delta, \texttt{app?}(s) \quad c, \Gamma \mid \texttt{---}\Delta, \texttt{abs?}(s), \texttt{app?}(s)}{\textbf{if } \texttt{app?}(s) \textbf{ then } a \textbf{ elsif } \texttt{abs?}(s) \textbf{ then } b \textbf{ else } c \textbf{ endif}, \Gamma \mid \texttt{---}\Delta}\ \texttt{(prop)}}{\textbf{cases } s \textbf{ of } \texttt{app}(x, t) : a, \texttt{abs}(t, u) : b, \texttt{vars}(t) : c \textbf{ endcases}, \Gamma \mid \texttt{---}\Delta}\ \texttt{(lift-if)}$$

The operational semantics of the binding instruction **let in** is related with the beta-contraction of the lambda calculus; thus, an expression of the form **let** $x = b$ **in** a has the interpretation $a[x/b]$, meaning that all instances of x in a will be simultaneously substituted by b, which in lambda calculus corresponds to the beta-contraction: $(\lambda_x.a)\ b \rightarrow_\beta a[x/b]$. **let in** instructions are interpreted in PVS by application of the `(beta)` command, but also automatically when commands such as `(assert)` are applied. For instance, consider the derivation below:

$$\dfrac{(a \textbf{ or not } b) \textbf{ or not } (a \textbf{ or not } b) \textbf{ and } a}{\Gamma \mid \texttt{---}\Delta, \textbf{let } d = (a \textbf{ or not } b) \textbf{ in } d \textbf{ or not } d \textbf{ and } a}\ \texttt{(assert)}$$

Chapter 5
Algebraic and Computational Examples

In this chapter, we present simple cases of study in order to illustrate the application of PVS for formalizing algebraic and algorithmic properties. The first example uses proof commands associated with Gentzen sequent and natural deduction rules to prove that an irrational to an irrational power might be rational and the second example uses also induction in order to prove correctness of sorting algorithms specified recursively. Formalizations related with examples in this section are available at the web page http://logic4CS.cic.unb.br.

5.1 Proving Simple Algebraic Properties

Initially, we will revisit the example that an irrational number to an irrational power maybe rational studied in Chap. 2 in the context of natural deduction.

We assume that the $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}}$ and $\sqrt{2}$ are respectively a rational and a irrational number, in order to provide deductions that there exist irrationals x and y such that x^y is a rational number. Derivations *à la* Gentzen Sequent Calculus and in Natural Deduction are given below. In these proofs, R denotes the unary predicate "rational" over numbers.

First, we present a derivation *à la* Gentzen Sequent Calculus. In this setting, our objective is to prove the sequent $\Rightarrow \exists x \exists y(\neg R(x) \wedge \neg R(y) \wedge R(x^y))$ and our assumptions are the sequents $\Rightarrow R((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})$ and $\Rightarrow \neg R(\sqrt{2})$.

The main rule of the proof is (Cut) using the sequent $\Rightarrow \neg R(\sqrt{2}^{\sqrt{2}}) \vee R(\sqrt{2}^{\sqrt{2}})$ which is easily obtained (namely, this is an instance of (LEM)).

Initially, proofs ∇_1 and ∇_2 of the sequents $\neg R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \exists x \exists y(\neg R(x) \wedge \neg R(y) \wedge R(x^y))$ and $R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \exists x \exists y(\neg R(x) \wedge \neg R(y) \wedge R(x^y))$ are given.

M. Ayala-Rincón and F.L.C. de Moura, *Applied Logic for Computer Scientists*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-51653-0_5

$$\nabla_1:\ \dfrac{\dfrac{\dfrac{}{\neg R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \neg R(\sqrt{2}^{\sqrt{2}})}\,(\mathrm{Ax}) \quad \dfrac{\dfrac{\Rightarrow \neg R(\sqrt{2})}{\neg R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \neg R(\sqrt{2})}\,(\mathrm{LW}) \quad \dfrac{\Rightarrow R((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})}{\neg R(\sqrt{2}^{\sqrt{2}}) \Rightarrow R((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})}\,(\mathrm{LW})}{\neg R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \neg R(\sqrt{2}) \wedge R((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})}\,(\mathrm{R}_\wedge)}{\neg R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \neg R(\sqrt{2}^{\sqrt{2}}) \wedge \neg R(\sqrt{2}) \wedge R((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})}\,(\mathrm{R}_\wedge)}{\neg R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \exists x \exists y (\neg R(x) \wedge \neg R(y) \wedge R(x^y))}\,(\mathrm{R}_\exists)^2$$

$$\nabla_2:\ \dfrac{\dfrac{\dfrac{\dfrac{\Rightarrow \neg R(\sqrt{2})}{R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \neg R(\sqrt{2})}\,(\mathrm{LW}) \quad \dfrac{\Rightarrow \neg R(\sqrt{2})}{R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \neg R(\sqrt{2})}\,(\mathrm{LW})}{R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \neg R(\sqrt{2}) \wedge \neg R(\sqrt{2})}\,(\mathrm{R}_\wedge) \quad \dfrac{}{R(\sqrt{2}^{\sqrt{2}}) \Rightarrow R(\sqrt{2}^{\sqrt{2}})}\,(\mathrm{Ax})}{R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \neg R(\sqrt{2}) \wedge \neg R(\sqrt{2}) \wedge R(\sqrt{2}^{\sqrt{2}})}\,(\mathrm{R}_\wedge)}{R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \exists x \exists y (\neg R(x) \wedge \neg R(y) \wedge R(x^y))}\,(\mathrm{R}_\exists)^2$$

And finally, we conclude by applying rule $(L_\vee)$ and (Cut).

$$\dfrac{(\mathrm{LEM})\ \Rightarrow \neg R(\sqrt{2}^{\sqrt{2}}) \vee R(\sqrt{2}^{\sqrt{2}}) \quad \dfrac{\nabla_1 \qquad \nabla_2}{\neg R(\sqrt{2}^{\sqrt{2}}) \vee R(\sqrt{2}^{\sqrt{2}}) \Rightarrow \exists x \exists y (\neg R(x) \wedge \neg R(y) \wedge R(x^y))}\,(\mathrm{L}_\vee)}{\Rightarrow \exists x \exists y (\neg R(x) \wedge \neg R(y) \wedge R(x^y))}\,(\mathrm{Cut})$$

The proof in natural deduction uses the assumptions $R((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})$ and $\neg R(\sqrt{2})$ and has as objective the formula $\exists x \exists y (\neg R(x) \wedge \neg R(y) \wedge R(x^y))$. The derivation uses (LEM) for $\neg R(\sqrt{2}^{\sqrt{2}}) \vee R(\sqrt{2}^{\sqrt{2}})$ and has as main rule $(\vee_e)$.

Initially, using the assumptions, we have natural derivations ∇'_1 and ∇'_2 for $\neg R(\sqrt{2}^{\sqrt{2}}) \vdash \exists x \exists y (\neg R(x) \wedge \neg R(y) \wedge R(x^y))$ and $R(\sqrt{2}^{\sqrt{2}}) \vdash \exists x \exists y (\neg R(x) \wedge \neg R(y) \wedge R(x^y))$.

$$\nabla'_1:\ \dfrac{\dfrac{[\neg R(\sqrt{2}^{\sqrt{2}})]^{a_1} \quad \dfrac{\neg R(\sqrt{2}) \quad R((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})}{\neg R(\sqrt{2}) \wedge R((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})}\,(\wedge_i)}{\neg R(\sqrt{2}^{\sqrt{2}}) \wedge \neg R(\sqrt{2}) \wedge R((\sqrt{2}^{\sqrt{2}})^{\sqrt{2}})}\,(\wedge_i)}{\exists x \exists y (\neg R(x) \wedge \neg R(y) \wedge R(x^y))}\,(\exists_i)^2$$

$$\nabla'_2:\ \dfrac{\dfrac{\dfrac{\neg R(\sqrt{2}) \quad \neg R(\sqrt{2})}{\neg R(\sqrt{2}) \wedge \neg R(\sqrt{2})}\,(\wedge_i) \quad [R(\sqrt{2}^{\sqrt{2}})]^{a_2}}{\neg R(\sqrt{2}) \wedge \neg R(\sqrt{2}) \wedge R(\sqrt{2}^{\sqrt{2}})}\,(\wedge_i)}{\exists x \exists y (\neg R(x) \wedge \neg R(y) \wedge R(x^y))}\,(\exists_i)^2$$

The proof is concluded as below:

$$\frac{\text{(LEM)}\ \ \neg R(\sqrt{2}^{\sqrt{2}}) \vee R(\sqrt{2}^{\sqrt{2}}) \qquad \nabla_1' \qquad \nabla_2'}{\exists x \exists y (\neg R(x) \wedge \neg R(y) \wedge R(x^y))}\ (\vee_e)\ a_1, a_2$$

Now, we examine a PVS deduction tree for proving this fact, specified as the conjecture below, where $R?$ abbreviates the predicate "rational" and when convenient, for a better visualization, the power operator, "^," is written in prefix notation (that is, $x \hat{} y$ is written as $\hat{}(x, y)$):

$$\textbf{exists}\,(x, y) : \textbf{not}\ R?(x)\ \textbf{and not}\ R?(y)\ \textbf{and}\ R?(\hat{}(x, y)).$$

The derivation tree is illustrated in Fig. 5.1. The root node is labelled by the objective sequent below, but for simplicity, all sequents were dropped from this tree:

$$|\text{---}\ \ \textbf{exists}\,(x, y) : \textbf{not}\ R?(x)\ \textbf{and not}\ R?(y)\ \textbf{and}\ R?(x \hat{} y).$$

As first derivation step, that is related with rule (cut), one must proceed by case analysis. For this, the command `(case)` is applied, and two branches are derived, as can be seen in Fig. 5.1. The left one labelled with a sequent which adds to the objective sequent the formula referent to the `(case)` as an antecedent and the right one with that formula as a succedent that are the sequents:

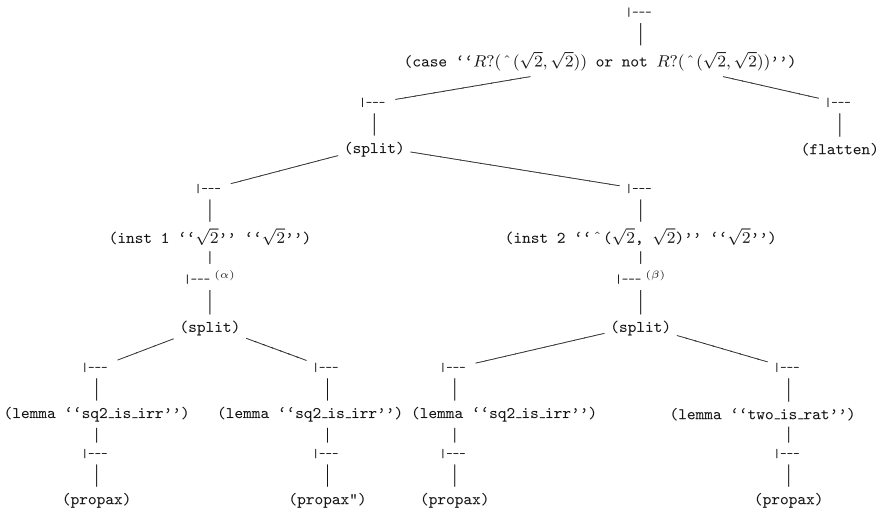


Fig. 5.1 Deduction PVS tree for an irrational to an irrational power maybe rational

$R?(\hat{}(\sqrt{2}, \sqrt{2}))$ **or not** $R?(\hat{}(\sqrt{2}, \sqrt{2}))$
|---
exists (x, y) **: not** $R?(x)$ **and not** $R?(y)$ **and** $R?(x \hat{} y)$,

and

|---
$R?(\hat{}(\sqrt{2}, \sqrt{2}))$ **or not** $R?(\hat{}(\sqrt{2}, \sqrt{2}))$
exists (x, y) **: not** $R?(x)$ **and not** $R?(y)$ **and** $R?(x \hat{} y)$.

The easy branch is the right one that is an instance of (LEM). This is almost automatically proved after using rule ($R_\vee$) that is applied through the PVS proof command `(flatten)`. Applying this command generates two succedent formulas, `R?(^(`$\sqrt{2}$`,`$\sqrt{2}$`))` and `NOT R?(^(`$\sqrt{2}$`,`$\sqrt{2}$`))`, and since c-equivalence moves the second formula to the antecedent without negation, the proof is concluded by automatic application of (Ax).

The left branch is the interesting one. The proof assistant provides the formula referent to the `(case)` as an antecedent. Thus, rule ($L_\vee$) should be applied to split the proof into two branches, which is done by application of the PVS proof command `(split)`. The proof tree is then split into two branches whose roots are labelled with the sequents (notice the use of c-equivalence in the second one):

$R?(\hat{}(\sqrt{2}, \sqrt{2}))$
|---
exists (x, y) **: not** $R?(x)$ **and not** $R?(y)$ **and** $R?(x \hat{} y)$,

and

|---
$R?(\hat{}(\sqrt{2}, \sqrt{2}))$
exists (x, y) **: not** $R?(x)$ **and not** $R?(y)$ **and** $R?(x \hat{} y)$.

The left sub-branch relates to both derivations ∇_2 and ∇'_2 and the right sub-branch to ∇_1 and ∇'_1. In both cases, one must deal adequately with the existential quantifiers in the target conjecture **exists** (x, y) **: not** $R?(x)$ **and not** $R?(y)$ **and** $R?(\hat{}(x, y))$. The Gentzen sequent rule to be applied is ($R_\exists$) that is applied through the PVS command `(inst)`. For the left sub-branch it should be applied the instantiation `(inst "`$\sqrt{2}$`" "`$\sqrt{2}$`")` that gives as result the sequent (labeling the node marked with (α) in Fig. 5.1).

$R?(\hat{}(\sqrt{2}, \sqrt{2}))$
|---
not $R?(\sqrt{2})$ **and not** $R?(\sqrt{2})$ **and** $R?(\hat{}(\sqrt{2}, \sqrt{2}))$.

The formula in the succedent splits into three objectives by Gentzen sequent rule ($R_\wedge$) through application of the PVS proof command `(split)`. The third of these objectives is trivially discharged by automatic application of (Ax), and the other two require the knowledge that **not** $R?(\sqrt{2})$, which is stated by a specific lemma called `sq2_is_irr`. These two branches are concluded by application of the PVS proof command `(lemma "sq2_is_irr")`.

For the right sub-branch, one applies the instantiation `(inst "^(√2, √2)" "√2")` obtaining the sequent (labeling the node (β) in Fig. 5.1).

$$
\begin{array}{l}
\vert\text{---} \\
R?(\hat{}(\sqrt{2}, \sqrt{2})) \\
\textbf{not}\ R?(\hat{}(\sqrt{2}, \sqrt{2}))\ \textbf{and not}\ R?(\sqrt{2})\ \textbf{and}\ R?(\hat{}(\hat{}(\sqrt{2}, \sqrt{2}), \sqrt{2})).
\end{array}
$$

After this, similarly to the previous branch, the PVS command `(split)` is applied obtaining three subobjectives. The first objective is automatically proved by applications of e-equivalence and (Ax). The other objectives are the sequents:

$$
\begin{array}{l}
R?(\sqrt{2}) \\
\vert\text{---} \\
R?(\hat{}(\sqrt{2}, \sqrt{2})),
\end{array}
$$

and

$$
\begin{array}{l}
\vert\text{---} \\
R?(\hat{}(\hat{}(\sqrt{2}, \sqrt{2}), \sqrt{2})) \\
R?(\hat{}(\sqrt{2}, \sqrt{2})).
\end{array}
$$

The former is proved by application of command `(lemma "sq2_is_irr")`, as done in the left sub-branch. The latter requires the knowledge that $R?(\hat{}(\hat{}(\sqrt{2}, \sqrt{2}), \sqrt{2})$ which is stated by a lemma called `two_is_rat` and applied with the command `(lemma "two_is_rat")`.

Previous two explained branches require the ability to provide adequate and concrete witnesses through instantiations as well as application of pertinent lemmas to cut proofs. In general, when dealing with quantifiers, the PVS proof command related with Gentzen sequent rules ($R_\exists$) and ($L_\forall$) is `(inst)`, while for both rules ($R_\forall$) and ($L_\exists$) what is required is application of Skolemization through the PVS proof command `(skolem)`. In the last case, the system will substitute quantified variables by *fresh* variables.

Exercise 61 Specify and prove that there exist irrationals such that one of them to the power of the other is rational. In PVS, you might assume that $\sqrt{2}$ is irrational through an axiom as given below:

```
ax1 :axiom not R?(sqrt(2)).
```

On the other side, $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}}$ can be proved equal to 2 expanding the operators ^ and `expt`, that are the operators related with the power operator.

5.2 Soundness of Recursive Algorithms Through Induction

To formalize correctness properties of algorithm, the focus on recursion and induction is necessary. As example of formalization, we propose very simple sorting algorithms. The NASA PVS library includes a complete theory developed by the authors called `sorting`, for a variety of sorting algorithms over non interpreted metric spaces with a well-founded measure [2]. Among others, the `sorting` theory includes soundness proofs of algorithms such as Mergesort, Quicksort, Heapsort, and Maxsort. Here, for simplicity, we use the type of naturals. As a simple instance, we use Hoare's Quicksort algorithm over naturals in this section.

Quicksort works as follows: empty lists are sorted and for a nonempty list l the problem is decomposed into sorting recursively the sublists l_1 and l_2 of elements that are, respectively, *less than or equal to* and *greater than* an element x of l, called *the pivot*. The sorted list is then obtained by appending the sorted list of the elements in l_1 with the pivot and the sorted list of the elements in l_2.

In general, quicksort is specified over lists of a noninterpreted type `T` in which a total measure is available. Lists are specified as usual as an inductive data structure where `null` is the constructor for empty lists, and `cons` constructs a new list from a given element of `T` and a list (see Exercise 57). As usual, the operators `cdr` and `car` give the tail and head of a list: `cdr(cons(x,l))` := `l` and `car(cons(x,l))` := `x`.

```
quicksort(l : list[T]) : recursive list[T] =
cases l
  | null : null,
  | cons(x, r) : append(quicksort(leq_elements(r, x) ),
                       cons(x,quicksort(g_elements(r, x))))
endcases
 measure length(l)
```

Algorithm 7: Specification of *quicksort* in PVS

In the above specification, the head of the list, `x`, is chosen as the pivot, and `leq_elements(r,x)` and `g_elements(r,x)` build the lists of elements of the tail `r` that are respectively less than or equal to and greater than `x`. See their specifications in the next two algorithms.

```
leq_elements(l : list[T], p : T): recursive list[T] =
case l
  | null : null,
  | cons (x, r):
  | if x <= p then
  |  | cons (x, leq_elements (r, p))
  | else
  |  | leq_elements (r, p)
  | end
endcases
 measure length(l)
```

Algorithm 8: Specification of the function `leq_elements` in PVS - list of elements less than or equal to a pivot

```
g_elements(l: list[T], p : T): recursive list[T] =
case l
  | null : null,
  | cons (x, r):
  | if x > p then
  |  | cons (x, g_elements (r, p))
  | else
  |  | g_elements (r, p)
  | end
endcases
 measure length(l)
```

Algorithm 9: Specification of the function `g_elements` in PVS - list of elements greater than a pivot

The termination measure for the function `quicksort` is given in the last line of its definition as the length of the list to be sorted. This measure is used to prove that `quicksort` is a well-defined function (and thus, that it is terminating). This is done proving that each recursive call of `quicksort` has, as argument, an *actual parameter* list whose length is strictly smaller than the input *parameter* list. In the easy cases, PVS can conclude well definedness of specified functions automatically, but in general, as happens in the case of `quicksort` above, the user needs to prove that the measure indeed decreases. The same happens for both functions `leq_elements` and `g_elements`.

For `quicksort`, well definedness is proved using the following lemmas:

```
leq_elements_size : LEMMA
   FORALL (l : list[T], x:T) :  length(leq_elements(l,x)) <= length(l)


g_elements_size : LEMMA
   FORALL (l : list[T], x:T) : length(g_elements(l,x)) <= length(l)
```

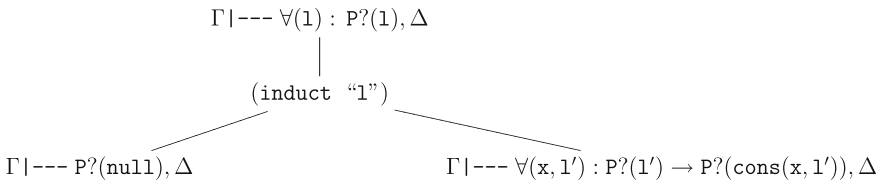


Fig. 5.2 Structural induction scheme for proving property `P?` on lists

The proofs of these lemmas require structural induction on lists. The structural induction principle of PVS is automatically generated by the command `(induct)` followed by the variable to be induced. In general, as discussed in Sect. 4.3, if `P?` is the predicate that represents the property on lists to be proved, starting from a sequent of the form $\Gamma \mid \texttt{---}\ \forall(\texttt{l}) : \texttt{P?}(\texttt{l}), \Delta$, then we have the induction scheme depicted in Fig. 5.2.

Exercise 62 Prove by structural induction the previous two lemmas.

Also, a complete or strong induction principle can be applied, as discussed in Sect. 4.3, where a different measure, say μ, on the inductive data structure (lists in our case), might be used. For doing this, the command `(measure-induct+)` is applied using as parameters the measure and the list of parameters required by the measure. Starting from a sequent of the form $\Gamma \mid \texttt{---}\ \forall(\texttt{l}) : \texttt{P?}(\texttt{l}), \Delta$, we have the induction scheme depicted in Fig. 5.3.

From the sequent $\Gamma \mid \texttt{---}\ \forall(\texttt{l}) : (\forall(\texttt{x}, \texttt{l}') : (\mu(\texttt{l}') < \mu(\texttt{l}) \rightarrow \texttt{P}(\texttt{l}'))) \rightarrow \texttt{P}(\texttt{l}), \Delta$, in Fig. 5.3, after applying proof commands `(skolem)` and `(flatten)` (associated respectively with SC rules $(R_\forall)$ and $(R_\rightarrow)$ as shown in Table 4.1) one has the sequent

$$\Gamma, (\forall(\texttt{x}, \texttt{l}') : (\mu(\texttt{l}') < \mu(\texttt{l}) \rightarrow \texttt{P}(\texttt{l}'))) \mid \texttt{---}\ \texttt{P}(\texttt{l}), \Delta,$$

which can also be obtained directly by the PVS command `(measure-induct+)` applied with the same arguments as above.

Using complete induction with measure `length(l)` on `l` to prove lemma `leq_elements_size` above, i.e., applying the proof command `(measure-induct+ "l")`, one obtains two subobjectives given below, where `leq_elements` is abbreviated as `leq_l`:

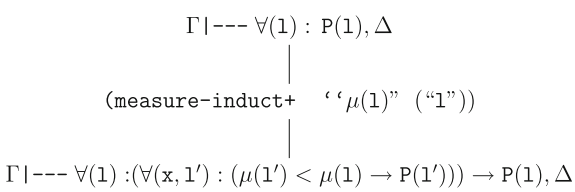


Fig. 5.3 Complete induction scheme for proving property `P?` on lists with measure μ on lists

```
|--- forall(x : T) : length(leq_l(null, x)), <= length(null)
```

and

```
|--- forall(x' : T, l' : list[T]) :
        forall(x : T) : length(leq_l(l', x)) <= length(l')
          implies
            forall(x : T) : length(leq_l(cons(x', l'), x)) <= length(cons(x', l'))
```

For proving the former, it is enough to apply the command `(skolem)` and then expand the definition of `leq_elements` obtaining the trivial sequent:

```
|--- length(null) <= length(null)
```

For proving the latter, after applying the commands `(skolem)` twice and then `(flatten)`, one obtains the subobjective:

```
forall(x : T) : length(leq_l(l', x)) <= length(l')
|---
forall(x : T) : length(leq_l(cons(x', l'), x)) <= length(cons(x', l'))
```

At this point application of proof commands related with rules ($R_\forall$) and ($L_\forall$) are required; indeed, first command `(skolem)` and then `(inst)` are applied. The latter with the adequate instantiation, obtaining the subobjective below:

```
length(leq_l(l', x)) <= length(l')
|---
length(leq_l(cons(x', l'), x)) <= length(cons(x', l'))
```

Expanding the definitions of `length` and `leq_elements` (i.e., `leq_l`), one obtains the following objective:

```
length(leq_l(l', x)) <= length(l')
|---
if x' <= x then 1 + length(leq_l(l', x))
else length(leq_l(l', x)) endif <= 1 + length(l')
```

Then, the instruction **if then else** in the succedent formula can be *lifted* by application of the proof command `(lift-if)`, obtaining the sequent below:

```
length(leq_l(l', x)) <= length(l')
|---
if x' <= x then 1 + length(leq_l(l', x)) <= 1 + length(l')
else length(leq_l(l', x)) <= 1 + length(l') endif
```

From this point, it is possible to apply the proof commands `(split)` and `(flatten)` obtaining two subobjectives, included below, that can be proved directly applying the command `(assert)` that, as previously discussed, will automatically apply the necessary simplifications from notions specified in the PVS prelude library using decision procedures.

```
x' <= x
length(leq_l(l', x)) <= length(l')
|---
1 + length(leq_l(l', x)) <= 1 + length(l')

length(leq_l(l', x)) <= length(l')
|---
x' <= x
length(leq_l(l', x)) <= 1 + length(l')
```

Translating complete PVS proofs to Sequent Calculus derivations is worth to consolidate the understanding of the relations between theory and practice. For illustrating this, consider the simple (auxiliary) property that after adding the same element to lists that are permutations one obtains lists that are also permutation, presented as the sequent below:

$$\Rightarrow \forall l_1, l_2, x\ (\texttt{Permutations?}(l_1, l_2) \rightarrow \texttt{Permutations?}(x \cdot l_1, x \cdot l_2))$$

Notice, `Permutations?` is a predicate that appears in the main correctness theorem of the Quicksort PVS theory given below:

```
quicksort_works: THEOREM    FORALL (l : list[T]):
  is_sorted?(quicksort(l)) AND Permutations?(quicksort(l), l)
```

For brevity, instead `Permutations?`, `Perm?` is used. This predicate is specified over pairs of lists based on the coincidence of occurrences of elements in both lists.

$$\texttt{Perm?}(l_1, l_2) : \texttt{Bool} = \textbf{forall}\, x\ :\ \texttt{Occurrences}(l_1, x) = \texttt{Occurrences}(l_2, x),$$

where $\texttt{Occurrences}(l, x)$ is specified as the number of occurrences of x in l, abbreviated as $\texttt{O}(l, x)$.

In order to provide a deduction *á la* Gentzen Sequent Calculus that prove that adding the same element x to permutations l_1 and l_2 one obtains lists $x \cdot l_1$ and $x \cdot l_2$ that are also permutations, assume the knowledge of the sequents:

$$Sq_A(l) \equiv\ \Rightarrow \forall y\, \texttt{O}(y \cdot l, y) = 1 + \texttt{O}(l, y),$$

and

$$Sq_B(l) \equiv \Rightarrow \forall x, y\, \neg x = y \rightarrow \mathtt{O}(x \cdot l, y) = \mathtt{O}(l, y),$$

where $x \cdot l$ abbreviates $\mathtt{cons}(x, l)$.

Below, a derivation *à la* Gentzen Sequent Calculus, ∇, is given that uses the extended transformation rules and follows the steps of the PVS proof (upside down).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\text{(LEM)} \Rightarrow x = y \vee \neg x = y} \quad \dfrac{\nabla_1 \qquad \nabla_2}{x = y \vee \neg x = y, \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(x \cdot l_1, y) = \mathtt{O}(x \cdot l_2, y)}\ (\mathrm{L}_\vee)}{\mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(x \cdot l_1, y) = \mathtt{O}(x \cdot l_2, y)}\ (\text{Cut})}{\forall y\, \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(x \cdot l_1, y) = \mathtt{O}(x \cdot l_2, y)}\ (\mathrm{L}_\forall)}{\forall y\, \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \forall y\, \mathtt{O}(x \cdot l_1, y) = \mathtt{O}(x \cdot l_2, y)}\ (\mathrm{R}_\forall)}{\mathtt{Perm?}(l_1, l_2) \Rightarrow \mathtt{Perm?}(x \cdot l_1, x \cdot l_2)}\ (\mathrm{L}_=)\ \text{by omitted def. of } \mathtt{Perm?}}{\Rightarrow \mathtt{Perm?}(l_1, l_2) \rightarrow \mathtt{Perm?}(x \cdot l_1, x \cdot l_2)}\ (\mathrm{R}_\rightarrow)}{\Rightarrow \forall x\, \mathtt{Perm?}(l_1, l_2) \rightarrow \mathtt{Perm?}(x \cdot l_1, x \cdot l_2)}\ (\mathrm{R}_\forall)}{\Rightarrow \forall l_2, x\, \mathtt{Perm?}(l_1, l_2) \rightarrow \mathtt{Perm?}(x \cdot l_1, x \cdot l_2)}\ (\mathrm{R}_\forall)}{\Rightarrow \forall l_1, l_2, x\, \mathtt{Perm?}(l_1, l_2) \rightarrow \mathtt{Perm?}(x \cdot l_1, x \cdot l_2)}\ (\mathrm{R}_\forall)$$

Where ∇_1 is given as

$$\dfrac{\dfrac{Sq_A(l_1) \qquad \nabla_1'}{\mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(y \cdot l_1, y) = \mathtt{O}(y \cdot l_2, y)}\ (\text{Cut})}{x = y, \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(x \cdot l_1, y) = \mathtt{O}(x \cdot l_2, y)}\ (\mathrm{L}_=)$$

with ∇_1'

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Rightarrow 1 + \mathtt{O}(l_1, y) = 1 + \mathtt{O}(l_1, y)}\ (\mathrm{R}_=)}{\mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow 1 + \mathtt{O}(l_1, y) = 1 + \mathtt{O}(l_2, y)}\ (\mathrm{L}_=)}{\mathtt{O}(y \cdot l_2, y) = 1 + \mathtt{O}(l_2, y), \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow 1 + \mathtt{O}(l_1, y) = \mathtt{O}(y \cdot l_2, y)}\ (\mathrm{L}_=)}{\forall y\, \mathtt{O}(y \cdot l_2, y) = 1 + \mathtt{O}(l_2, y), \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow 1 + \mathtt{O}(l_1, y) = \mathtt{O}(y \cdot l_2, y)}\ (\mathrm{L}_\forall)}{\mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow 1 + \mathtt{O}(l_1, y) = \mathtt{O}(y \cdot l_2, y)}\ (\text{Cut})}{\mathtt{O}(y \cdot l_1, y) = 1 + \mathtt{O}(l_1, y), \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(y \cdot l_1, y) = \mathtt{O}(y \cdot l_2, y)}\ (\mathrm{L}_=)}{\forall y\, \mathtt{O}(y \cdot l_1, y) = 1 + \mathtt{O}(l_1, y), \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(y \cdot l_1, y) = \mathtt{O}(y \cdot l_2, y)}\ (\mathrm{L}_\forall)}$$

$Sq_A(l_2)$

And, where ∇_2 is given as

$$\dfrac{\begin{array}{c} Sq_B(l_1) \\ \vdots \\ \neg x = y \Rightarrow \mathtt{O}(x \cdot l_1, y) = \mathtt{O}(l_1, y) \qquad \nabla_2' \end{array}}{\neg x = y, \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(x \cdot l_1, y) = \mathtt{O}(x \cdot l_2, y)} \text{ (Cut)}$$

with ∇_2'

$$\dfrac{\dfrac{\begin{array}{c} Sq_B(l_2) \\ \vdots \\ \neg x = y \Rightarrow \mathtt{O}(x \cdot l_2, y) = \mathtt{O}(l_2, y) \end{array} \qquad \dfrac{\mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(l_1, y) = O(l_2, y) \text{ (Ax)}}{\mathtt{O}(x \cdot l_2, y) = \mathtt{O}(l_2, y), \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(l_1, y) = \mathtt{O}(x \cdot l_2, y)} \text{ (L}_=\text{)}}{\neg x = y, \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(l_1, y) = \mathtt{O}(x \cdot l_2, y)} \text{ (cut)}}{\mathtt{O}(x \cdot l_1, y) = \mathtt{O}(l_1, y), \neg x = y, \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y) \Rightarrow \mathtt{O}(x \cdot l_1, y) = \mathtt{O}(x \cdot l_2, y)} \text{ (L}_=\text{)}$$

Exercise 63 a. In the main derivation above, that is, ∇ complete the step labelled as "(L$_=$) by omitted definition of `Permutation?`." Notice that rule (L$_=$) is being used with equation $\mathtt{Perm?}(l_1, l_2) = \forall y\, \mathtt{O}(l_1, y) = \mathtt{O}(l_2, y)$.

b. Complete the steps in the derivation below used in ∇_2 and ∇_2'.

$$\dfrac{\begin{array}{c} Sq_B(l) \equiv \Rightarrow \forall x, y\, \neg x = y \rightarrow \mathtt{O}(x \cdot l, y) = \mathtt{O}(l, y) \\ \vdots \end{array}}{\neg x = y \Rightarrow \mathtt{O}(x \cdot l, y) = \mathtt{O}(l, y)}$$

Now we will give a PVS proof for the sequent:

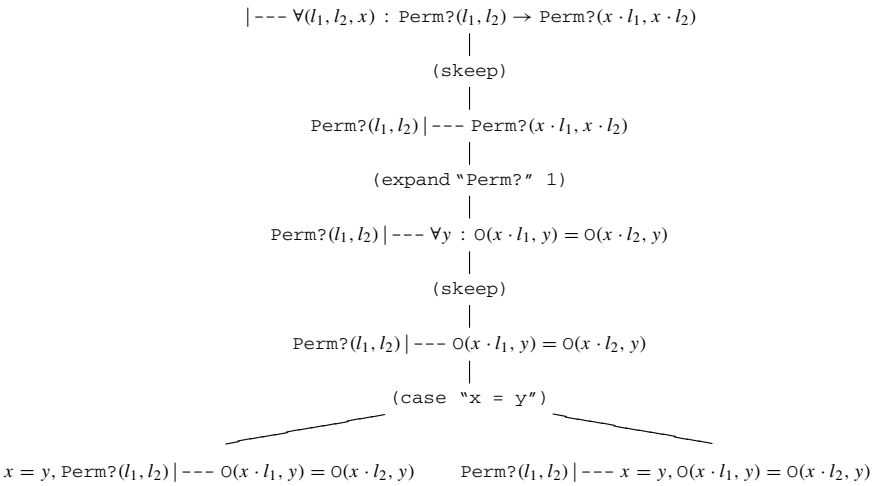
```
|--- forall(l1, l2 : list[T], x : T) : Permutations?(l1, l2) → Permutations?(x · l1, x · l2).
```

Instead the sequents Sq_A and Sq_B, used in the derivation *à la* Gentzen Sequent Calculus, we use the lemmas below that are easy to be proved from the definition of `Occurrences`.

```
L_SqA : LEMMA
   FORALL (l : list[T], y : T) : O(y.l,y)) = 1 + O(l,y)

L_SqB : LEMMA
   FORALL (l : list[T], x, y :T) : NOT x = y -> O(x.l,y)) = O(l,y)
```

The proof is formalized as explained below. From the objective, after a few command applications we split the proof in two cases.



The left branch is formalized as below:

$$
\begin{array}{c}
x = y, \texttt{Perm?}(l_1, l_2) \mid \texttt{---}\ \texttt{O}(x \cdot l_1, y) = \texttt{O}(x \cdot l_2, y) \\
\texttt{(replaces -1)} \\
\texttt{Perm?}(l_1, l_2) \mid \texttt{---}\ \texttt{O}(y \cdot l_1, y) = \texttt{O}(y \cdot l_2, y) \\
\texttt{(lemma "L\_SqA")} \\
\forall(l, y) : \texttt{O}(y \cdot l, y) = 1 + \texttt{O}(l, y), \texttt{Perm?}(l_1, l_2) \mid \texttt{---}\ \texttt{O}(y \cdot l_1, y) = \texttt{O}(y \cdot l_2, y) \\
\texttt{(inst -1 "}l_1\texttt{" "y")} \\
\texttt{O}(y \cdot l_1, y) = 1 + \texttt{O}(l_1, y), \texttt{Perm?}(l_1, l_2) \mid \texttt{---}\ \ \texttt{O}(y \cdot l_1, y) = \texttt{O}(y \cdot l_2, y) \\
\texttt{(replaces -1)} \\
\texttt{Perm?}(l_1, l_2) \mid \texttt{---}\ 1 + \texttt{O}(l_1, y) = \texttt{O}(y \cdot l_2, y) \\
\texttt{(lemma "L\_SqA"); (inst -1 "}l_2\texttt{" "y");} \\
\texttt{(replaces -1)} \\
\texttt{Perm?}(l_1, l_2) \mid \texttt{---}\ 1 + \texttt{O}(l_1, y) = 1 + \texttt{O}(l_2, y) \\
\texttt{(expand "Perm?")} \\
\forall(y) : \texttt{O}(l_1, y) = \texttt{O}(l_2, y) \mid \texttt{---}\ 1 + \texttt{O}(l_1, y) = 1 + \texttt{O}(l_2, y) \\
\texttt{inst?}
\end{array}
$$

The right branch is formalized in a similar manner as depicted below:

$$\texttt{Perm?}(l_1, l_2) \,|\texttt{---}\; x = y, \texttt{O}(x \cdot l_1, y) = \texttt{O}(x \cdot l_2, y)$$
$$|$$
$$\texttt{(lemma "L\_SqB")}$$
$$|$$
$$\forall(l, x, y) : \neg x = y \to \texttt{O}(x \cdot l, y) = \texttt{O}(l, y), \texttt{Perm?}(l_1, l_2) \,|\texttt{---}\; x = y, \texttt{O}(x \cdot l_1, y) = \texttt{O}(x \cdot l_2, y)$$
$$|$$
$$\texttt{(inst -1 "}l_1\texttt{" "x" "y")}$$
$$|$$
$$\neg x = y \to \texttt{O}(x \cdot l_1, y) = \texttt{O}(l_1, y), \texttt{Perm?}(l_1, l_2) \,|\texttt{---}\; x = y,\ \ \texttt{O}(x \cdot l_1, y) = \texttt{O}(x \cdot l_2, y)$$
$$|$$
$$\texttt{(assert); (replaces -1)}$$
$$|$$
$$\texttt{Perm?}(l_1, l_2) \,|\texttt{---}\; x = y,\ \ \texttt{O}(l_1, y) = \texttt{O}(x \cdot l_2, y)$$
$$|$$
$$\texttt{(lemma "L\_SqB"); (inst -1 "}l_2\texttt{" "x" "y"); (assert); (replaces -1)}$$
$$|$$
$$\texttt{Perm?}(l_1, l_2) \,|\texttt{---}\; x = y,\ \ \texttt{O}(l_1, y) = \texttt{O}(l_2, y)$$
$$|$$
$$\texttt{(expand "Perm?")}$$
$$|$$
$$\forall(y) : \texttt{O}(l_1, y) = \texttt{O}(l_2, y) \,|\texttt{---}\; 1 + \texttt{O}(l_1, y) = 1 + \texttt{O}(l_2, y)$$
$$|$$
$$\texttt{inst?}$$

The main correctness Theorem for Quicksort requires proving that the computed output is indeed a permutation of the input list of naturals, `l`. This proof starts with the sequent below:

```
|--- FORALL (l : list[T]): Perm?(quicksort(l),l)
```

In Fig. 5.4, we present a sketch of the PVS formalization in which a few nodes that are labelled with sequents are abbreviated with labels of the form $\vdash$ and $\vdash^{(n)}$. The root node of the derivation tree with label $^{(0)}$ is for the target sequent above.

The proof proceeds by strong induction applying the command `(measure-induct+)` using as measure function the length of lists (see proof sketch in Fig. 5.4). Examining proofs such as this one, important proof strategies such as multiple use of the induction hypothesis can be highlighted. This happens in the formalization of soundness of Quicksort, both for proving that the output is in fact a sorted list and a permutation of the input list. More specifically, the induction hypothesis has to be applied to two sublists that are shorter than the input list: the list of elements less than or equal to the pivot and the list of those greater than the pivot. After application of the induction command one obtains the sequent below (depicted by short as $\vdash^{(1)}$ in Fig. 5.4), where the induction hypothesis is the antecedent formula (type annotations are omitted by short).

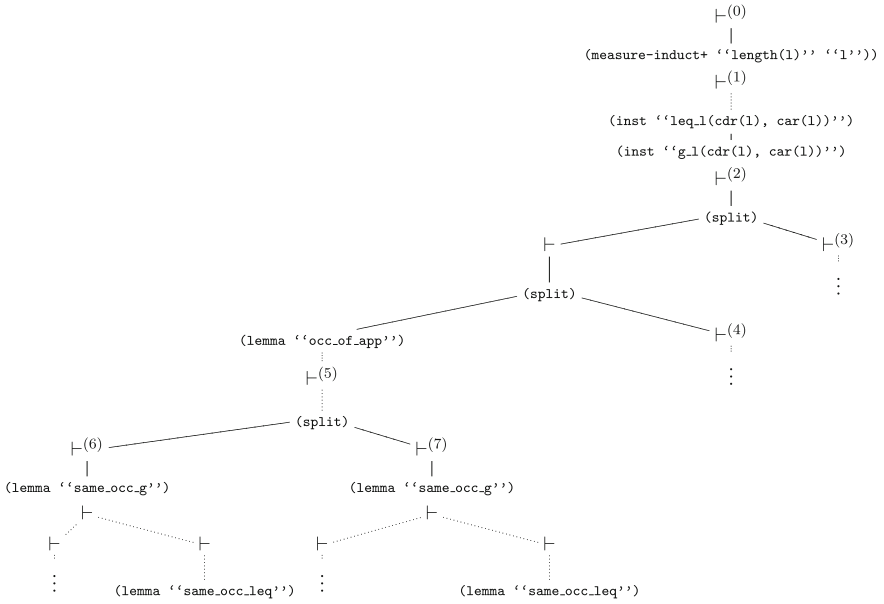


Fig. 5.4 Deduction PVS tree to permutation of Quicksort

```
FORALL(y): length(l') < length(l) IMPLIES Perm?(quicksort(l'),l')
|--- Perm?(quicksort(l),l)
```

After that, proper applications of `(expand)` are done (that are related to the derivation rule ($L_{=}$)) for expanding the definitions of `quicksort`, `Perm?`, `O`, etc. Additionally, the proof command `(copy)` (that is related to (LW)) is applied in order to duplicate the induction hypothesis, and the command `(inst)` is applied with adequate parameters in order to apply the induction hypothesis twice: one for the list of elements that are less than or equal to the pivot and other for the list of elements that are greater than the pivot. These instantiations are related with the application of the derivation rule ($L_{\forall}$). After all that, one obtains the following sequent (depicted as $\vdash^{(2)}$ in Fig. 5.4), where for brevity in addition to the abbreviation `leq_l`, we use also `q_s`, `g_l` and `app` for abbreviating, respectively, `quicksort`, `g_elements` and `append`:

```
  length(g_l(cdr(l), car(l))) < length(l) IMPLIES
     FORALL (x):O(q_s(g_l(cdr(l), car(l))), x) = O(g_l(cdr(l), car(l)), x),
  length(leq_l(cdr(l), car(l))) < length(l) IMPLIES
     FORALL (x):O(q_s(leq_l(cdr(l), car(l))), x) = O(leq_l(cdr(l), car(l)), x)
|---
  null?(l),
  O(app(q_s(leq_l(cdr(l), car(l))),
        cons(car(l), q_s(g_l(cdr(l), car(l))))), x) = O(l, x)
```

At this point of the formalization, after splitting the antecedent formulas of this sequent, and applying `(inst "x")` twice (see applications of `(split)` in Fig. 5.4), several simple lemmas should be applied.

- For the right branches generated by these two application of the proof command `(split)`, that are the branches rooted by $\vdash^{(3)}$ and $\vdash^{(4)}$, the proof command `(lemma)` is used to apply both lemmas `leq_elements_size` and `g_elements_size` mentioned at the begin of this section. This is done to prove correct instantiation of the inductive hypotheses with lists that are in fact shorter than the input list `l`, that are the lists `g_l(cdr(l), car(l))` and `leq_l(cdr(l), car(l))` of elements greater than and less than or equal to the pivot, respectively.
- For the left branch generated by these two applications of `(split)`, lemmas `occ_of_app`, `same_occ_leq` and `same_occ_g` are applied, whose meaning will be explained below.

The lemma `occ_of_app` states the simple fact that the occurrences of some element in the append of two lists is the sum of the occurrences of the element in the lists being appended.

```
occ_of_app : LEMMA
   FORALL (l1, l2 : list[T], x : T) : O(app(l1,l2),x) =  O(l1,x) + O(l2,x)
```

Applying this lemma, by using commands `(lemma)` and `(inst)`, the latter with the proper instantiations, one adds the following antecedent formula to the previous sequent (the new sequent corresponds to node $\vdash^{(5)}$ in Fig. 5.4).

```
  O(app(q_s(leq_l(cdr(l), car(l))),
    cons(car(l), q_s(g_l(cdr(l), car(l))))), x)
=    O(q_s(leq_l(cdr(l), car(l))), x) +
     O(cons(car(l), q_s(g_l(cdr(l), car(l)))), x)
```

Then, the left-hand side of this equation is replaced by the right-hand side in the succedent of the sequent (applying command `(replaces)`) obtaining a sequent with the modified formula of interest in the succedent.

```
  ...
|---
  ...
  O(q_s(leq_l(cdr(l), car(l))), x) +
  O(cons(car(l), q_s(g_l(cdr(l), car(l)))), x) = O(l, x)
```

Then, `Occurrences` (i.e., `O`) is expanded at the second occurrence of `O` in the second formula in the succedent. Since the expanded definition considers whether `car(l) = x` or not, after application of the commands `(lift-if)` and `(split)`, two cases are to be considered according to whether `car(l) = x` or `NOT car(l) = x`. These cases correspond to branches rooted by ($\vdash^{(6)}$) and ($\vdash^{(7)}$) in Fig. 5.4.

In order to conclude, one proceeds by application of lemmas `same_occ_leq` and `same_occ_g` that are used to relate occurrences of elements in both lists `leq_l(cdr(l), car(l))` and `g_l(cdr(l), car(l))`. These lemmas are specified as below:

```
same_occ_leq: LEMMA
  FORALL (l:list[nat], x, p : nat):
    x <= p IMPLIES O(l,x) = O(leq_l(l,p), x) AND O(g_l(l,p))(x) = 0

same_occ_g: LEMMA
  FORALL (l:list[nat], x, p : nat):
    x > p  IMPLIES O(l,x) = O(g_l(l,p),x) AND O(leq_l(l,p), x) = 0
```

The former lemma states that for an element `x` that is less than or equal to the pivot `p`, its occurrences in the lists `l` and `leq_l(l, p)` are equal, while it does not occur in the list `g_l(l, p)`. Similarly, the latter lemma states that for an element `x` that is greater than `p`, its occurrences in the lists `l` and `g_l(l, p)` are equal, while it occurs zero times in the list `leq_l(l, p)`. These lemmas should be applied with the adequate instantiation, that is, using as list the tail `cdr(l)` and as pivot the head `car(l)`, of the input list `l`.

Exercise 64

a. Complete all details of this formalization. For doing this you could also check directly the formalization of correctness of quicksort in the theory for sorting algorithms.
b. Build a derivation *à la* Gentzen Sequent Calculus for this formalization and compare with the proof steps in item *a.*, correspondingly with the proof steps sketched in Fig. 5.4.

Chapter 6
Suggested Readings

6.1 Proof Theory

Proof theory is a subarea of mathematical logic that deals with proofs as mathematical objects. In principle, our focus is on the so called *structural proof theory*, that is a subbranch of proof theory devoted to the study of properties of deductive calculi such as natural deduction and sequent calculus. The German mathematician Gerhard Gentzen submitted his famous manuscript in 1933, in which he introduced the calculus of natural deduction and the sequent calculus for both the intuitionistic and classical logics. But also, it is known that the Polish logician Stanisław Jaśkowski, who was a descendant from Jan Łukasiewicz, independently developed in 1929 his own calculus of natural deduction. In Gentzen's pioneering work, entitled *Untersuchungen über das logische Schließen* and published in two parts only in 1935 [10, 11], he formalized his *Kalkül des natürlichen Schließens* for the intuitionistic (NJ) and the classical (NK) predicate logic as well as sequent calculi for the intuitionistic (LJ) and the classical (LK) logic. These calculi correspond to the calculi examined in this work. Gentzen used the sequent calculi to provide as main result the theorem of cut elimination and then the equivalence between natural and sequent calculi.

The mathematical aspects of deductive systems can be studied in excellent textbooks on proof theory as for instance Troelstra and Schwichtenber's one [34], Negri and von Plato [23], or Prawitz' 1960s classical one [30] on systems of natural deduction. Also, complete presentations of proof theory are available as the one by Aczel, Simmons and Wainer [1], and more modern presentations as those by Schwichtenber and Wainer [32] and Pohlers [28]. An encyclopedic version is edited by Buss [4] and philosophical aspects are discussed by Hendricks, Pedersen, and Jorgensen in [14]. The first two chapters of [9] bring an excellent mathematical introduction in proof theory. Also, accessible English translations of Gentzen seminal papers [10,11] are available.

Structural proof theory appeared as an adequate formal treatment to answer the German mathematician David Hilbert's famous program: to prove the consistency of mathematics by consistent and reliable but simpler foundational methods. Although

M. Ayala-Rincón and F.L.C. de Moura, *Applied Logic for Computer Scientists*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-51653-0_6

Hilbert's program was answered negatively by the Austrian logician Kurt Gödel in 1931 [13]—and subsequently other negative answers to the so called *Entscheidungsproblem*, were given, such as Alan Turin's well-known theorem on the undecidability of the halting problem (straightforward consequence of results originally given in [36]) as well as Alonzo Church's unsolvability presentation in the context of lambda calculus [5]—proof theory remains of great importance and influence in computer science providing formal frameworks for the development of automated reasoning tools.

6.2 Formal Logic

Despite in the current work only the very basics of logic for computer science is covered, and with a specialized focus on one relevant application of logic in formal deduction, the reader should be advised of essential nice and crucial results in formal logic that deserve his/her attention. From the formal foundational results the most elaborated ones presented here are related with the soundness and completeness of the deductive calculi. Thus, the importance of having included a sketch of Gödel's completeness theorem that relates semantic truth and formal deduction. Gödel's completeness theorem was originally given in his Thesis presented at the University of Vienna and entitled *Über die Vollständigkeit des Logikkalküls* and further published in 1930 as [12]. The proof sketch given here follows the simplified model theory style introduced by the U.S. American logician Leon Albert Henkin in 1949 in [15], which is the standard one presented in modern textbooks on mathematical logic.

Post [29] proved the undecidability of the word problem for Thue systems that essentially are monoids. Post's proof consists in the reduction of the halting problem in a Turing machine to an instance of the word problem in a related monoid. This work is known as the first unsolvability result for a decision problem of classical mathematics. The equality for this instance of the word problem holds if and only if the Turing machine halts. Hence, by the undecidability of the halting problem for Turing machines the word problem for these structures should be undecidable in general. Grigorii Samuilovich Tseitin's concrete monoid for which the word problem is known undecidable and that was used to prove the undecidability of validity of the predicate logic in Chap. 2 was published almost ten years after Post's paper on the undecidability of the word problem for monoids [35].

Logic is related with computability, decidability, and undecidability of problems that were not thoroughly revised in this book and usually are only studied in all details in advanced courses on mathematical or formal logic. Among the fundamental results omitted (or just referenced) in this book we would mention a few cornerstone theorems related with the computational expressiveness and limits of the first-order logic.

The Löwenheim-Skolem's theorem states that if a countably first-order theory has an infinite model, then it has models of any infinite cardinality. This implies that it is not possible to achieve first-order specifications of mathematical structures such

as naturals, rationals, reals, etc., since every attempt to specify such structures will allow models with arbitrary (higher or lower) infinite cardinality that consequently would not be isomorphic to the target structure. The compactness theorem states that a set of first-order sentences is satisfiable if and only if every finite subset of it is satisfiable. This property allows the constructions of models of any set of sentences that is finitely consistent. The countable case was proved by Gödel, while the Russian mathematician Anatoly Maltsev proved the uncountable case in 1936. Both the Löwenheim-Skolem and the compactness theorems are crucial for the minimal characterization of first-order logic expressed as the Lindström's theorem, published in 1969 [20] and named after the Swedish logician Per Lindström. This theorem states that first-order logic is the strongest logic holding the compactness property and the (downward) Löwenheim-Skolem's property. As a consequence one has that any possible extension of first-order logic, built for instance to specify algebraic structures such as naturals, reals, etc., will loss at least one of these properties.

Of course, the study of Gödel's incompleteness theorems is also relevant. Gödel paramount incompleteness theorems show that there are limitations in what can be achieved with formal mathematical proofs. The first incompleteness theorem informally states that "every sufficient rich and consistent axiom system contains meaningful statements that are undecidable within the system itself," it was announced by Gödel during the *Second Conference for Epistemology of the Exact Sciences* on September the seventh, 1930. The significance of this result to Hilbert's program was immediately understood by some of the attendants. In particular, the Hungarian mathematician John von Neumann, who was working on Hilbert's program at that time, was attending this meeting. According to von Neumann's letters to the German philosopher Rudolf Carnap and to Gödel himself, it appears to be that von Neumann inferred the second incompleteness theorem [31]. The second incompleteness theorem states that "the consistency of a sufficiently rich axiomatic theory, such as Peano Arithmetic, cannot be proved within the system itself, i.e., that the statement expressing the consistency of the system is undecidable in the system." Von Neumann wrote a letter to Gödel with a sketch of his ideas in November the 20th, and a second one the 29th that month, in which he thanks Gödel's reprint (answer to the first letter) and also wrote "Since you have established the theorem on the unprovability of consistency as a natural continuation and deepening of your earlier results, I clearly won't publish on this subject." In a letter to Rudolf Carnap dated June 7th, 1931, von Neumann wrote "Gödel has shown the unrealizability of Hilbert's program," but also that "there is no more reason to reject intuitionism," point in which he disagreed with Gödel, who thought his results do not contradict Hilbert's formalistic viewpoint. Gödel incompleteness theorems were originally published in 1931 [13].

Formal and mathematical aspects of logic can be consulted in excellent textbooks from which we give only a small list. Ebbinghaus, Flum, and Thomas' textbook brings a complete and nice presentation of all results previously mentioned [8]. This book includes a nice chapter on the resolution principle used for the implementation of logical programming languages. Also Enderton's classical book (originally published in 1972) brings a nice precise mathematical presentation of the basic results to understand the incompleteness theorems [9], and Huth and Ryan's book [17] offers

a presentation directed to computer science in which interesting applications such as SAT solvers, program verification, and model checking are covered.

6.3 Automated Theorem Proving and Type Theory

Two of the main branches of structural proof theory are automated theorem proving and type theory. Automated theorem proving gave rise to a myriad of *type-theoretical* deductive tools, automated theorem provers, and proof assistants, perhaps all of them pioneered by the Dutch mathematician Nicolaas Govert de Bruijn's group work on the system Automath [21]. This system was developed at Eindhoven in the 1960s and 1970s with the aim of formalizing mathematics and particularly the corpus of mathematical knowledge in Edmund Landau's textbook *Grundlagen der Analysis*, which is a nice and rigorous piece of formal work by itself, written in the late 1920s and that has been influenced modern presentations on the fundamentals of mathematical analysis until today. Among the available modern deductive computational frameworks, we can highlight PVS, Coq, Isabelle, HOL, Matita, Lean (L∃∀n), Agda, and ACL2. Type theory not only provides the fundamental framework for the development of the deductive engine of proof assistants, but in general for the development of robust computational languages in which errors could be detected before computation.

Types were introduced in 1910 by Alfred North Whitehead and Bertrand Russell in their *Principia Mathematica* to avoid inconsistencies in the foundations of mathematics. Computational frameworks such as Alonzo Church's lambda calculus were enriched with types [6] as a mechanism to provide robust formalisms in which paradoxes are avoided. Basic simple types in the lambda calculus provided the technological basis of programming languages with types in the 1970s. The American and Belgian logicians Haskell Curry and Robert Feys devised in the late 1950s [7] the type inference algorithm embedded in the ML family of programming languages developed by Robin Milner's group at Edinburgh University in the early 1980s. The formal aspects of simply typed lambda calculus are nicely presented in textbooks such as Hindley's one [16] and Kamareddine, Laan, and Nederpelt's one [18]. The authors of the latter book, also exposed in detail the early history of the development of the theory in [19]. Applications of type theory in computation are exhaustively exposed in Pierce's books on types and programming languages [26, 27], and an encyclopedic presentation of the mathematical formalisms of modern type systems in the lambda calculus is given by Barendregt, Dekkers, and Statsman in [3]. The latter book covers simple, recursive, and intersection types. Nederpelt and Gouvers' book [22] covers on its side, higher-order, inductive, and dependent types, including also the well-known *calculus of constructions*, with focus on the application of type systems for the representation and verification of mathematical knowledge.

Recently, the Russian mathematician Vladimir Voevodsky's program *"Univalent" Foundations of Mathematics* arose with a lot of momentum in the community of mathematicians with the objective of having a comprehensive, computational foundation for mathematics based on the homotopical interpretation of type theory

[33]. This program aroused great interest in the research community in part by the influence of Voevodsky, who won the Fields Medal in 2002 for his work on homotopy theory for algebraic varieties. The program is being developed in the calculus of inductive constructions, the formalism behind the proof assistant Coq. Homotopy type theory develops intentional type theory using type theory as a language to formulate mathematics within a type-theoretical foundation.

Most mentioned deductive computational tools are implemented in the basis of some type system. PVS uses an extension of Church's theory of types with dependent types. Lean, as PVS, is based on dependent type theory. Agda, is a dependently typed functional programming language in which proofs are written in a functional style as well. Coq uses the famous calculus of constructions that is a higher-order typed lambda calculus created by Thierry Coquand. Matita is based on a dependent type system known as the calculus of coinductive constructions, which is a derivative of Coquand's calculus of constructions. Isabelle is implemented in Standard ML, thus inheriting the ML treatment of types, through a weak theory of types used as a metalogic, which encodes object logics such as first-order logic (FOL), higher-order logic (HOL), etc. Theorems are objects of a specific abstract data type and the type system ensures that only adequate inference rules are applied in the derivation of a theorem. HOL is an acronym for *Higher-Order Logic* that as Isabelle, is a successor of the well-known theorem prover LCF which incorporates ML and uses the same principles. Indeed, HOL is a family of higher-order proof assistants (HOL4, HOL Light, HOL Zero, etc.). HOL uses a higher-order predicate logic with terms from the Church's simply typed lambda calculus. The system of types includes as usual type variables, atomic types, and function types, but also compound types which are type operators for constructing sets from sets such as the type of lists of elements of some set, the Cartesian product of sets, etc. ACL2 is an acronym for *A Computational Logic for Applicative Common Lisp*, which is a functional programming language in which not only Lisp's style implementations can be specified, but also properties about them can be proved. ACL2 uses five datatypes of the Common Lisp programming language: numbers, characters, strings, symbols, and conses, which are lists or trees. In fact, ACL2 is not based on any type system and essentially what is allowed is that objects of these kinds can be passed as values to functions, return them as outputs of functions, and store them in lists and trees. ACL2 provides operators for each type and no operator can modify the objects passed in as arguments.

Except for PVS no references related with these proof assistants are included here (see Chap. 4), since the readers will easily find in the web a great variety of updated tutorials, manuals, scientific and technical articles, and even books, and the most important, will be able to download and install the tools and perform their own experiments.

References

1. Aczel, P., Simmons, H., Wainer, S. (eds.): Proof Theory. Cambridge University Press (1992)
2. Ayala-Rincón, M., Almeida, A.A., Ramos, T.M.F., de Moura, F.L.C., Rocha-Oliveira, A.C.: PVS `sorting` theory. Available at http://logic4CS.cic.unb.br and as part of the LaRC Formal Methods NASA PVS libraries at http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library, Sept (2016)
3. Barendregt, H.P., Dekkers, W., Statman, R.: Lambda Calculus with Types. Perspectives in Logic, Cambridge University Press (2013)
4. Buss, S.R. (ed.): Handbook of Proof Theory. Studies in Logic, vol. 137. Elsevier, North Holland (1998)
5. Church, A.: An unsolvable problem of elementary number theory. Am. J. Math. **58**(2), 345–363 (1936)
6. Church, A.: A formulation of the simple theory of types. J. Symbolic Logic **5**, 56–68 (1940)
7. Curry, H.B., Feys, R.: Combinatory Logic, vol. 1. North Holland (1958)
8. Ebbinghaus, H-D., Flum, J., Thomas, W.: Mathematical logic. Undergraduate texts in mathematics, 2nd edn. Springer (1996)
9. Enderton, H.B.: A Mathematical Introduction to Logic, 2nd edn. Academic Press Inc. (2001)
10. Gentzen, G.: Untersuchungen über das logische Schließen I. Mathematische Zeitschrift **39**, 176–210 (1935)
11. Gentzen, G.: Untersuchungen über das logische Schließen II. Mathematische Zeitschrift **39**, 405–431 (1935)
12. Gödel, K.: Die Vollständigkeit der Axiome des logischen Funktionenkalkülus. Monatshefte für Mathematik und Physik **37**(1), 349–360 (1930)
13. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. I. Monatshefte für Mathematik und Physik **38**, 159–166 (1931)
14. Hendricks, V.F., Pedersen, S.A., Jorgensen, K.F. (eds.): Proof Theory—History and Philosophical Significance, volume 292 of Synthese Library. Kluwer Academic Publishers (2010)
15. Henkin, L.: The completeness of the first-order functional calculus. J. Symb. Log. **14**(3), 159–166 (1949)
16. Hindley, J.R.: Basic Simple Type Theory. Cambridge University Press (1997)
17. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems, 2nd edn. Cambridge University Press (2004)
18. Kamareddine, F.D., Laan, T., Nederpelt, R.: A Modern Perspective on Type Theory. Number 29 in Applied Logic Series. Kluewer (2004)
19. Kamareddine, F.D., Laan, T., Nederpelt, R.: A History of types*. In: Logic: A History of its Central Concepts, volume 11 of Handbook of the History of Logic, pp. 451–511. Elsevier (2012)

M. Ayala-Rincón and F.L.C. de Moura, *Applied Logic for Computer Scientists*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-51653-0

20. Lindström, P.: On Extensions of Elementary Logic. Theoria, pp. 1–11 (1969)
21. Nederpelt, R.P., Geuvers, J.H., de Vrijer, R.C.: Selected papers on Automath. North-Holland (1994)
22. Nederpelt, R., Geuvers, H.: Type Theory and Formal Proof—An Introduction. Cambridge University Press (2014)
23. Negri, S., von Plato, J.: Structural Proof Theory, 1st edn. Cambridge University Press (2001)
24. Owre, S., Shankar, N.: The formal semantics of PVS. In: Technical Report SRI-CSL-97-2, Computer Science Laboratory, SRI International, Menlo Park, CA, August 1999. Revised version of NASA LaRC Contract Report NASA/CR-1999-209321 (1997)
25. Owre, S., Shankar, N.: A brief overview of PVS. In Theorem Proving in Higher Order Logics, TPHOLs 2008, volume 5170 of Lecture Notes in Computer Science, pp. 22–27. Springer (2008)
26. Pierce, B.C.: Types and Programming Languages. MIT Press (2002)
27. Pierce, B.C. (ed.): Advanced Topics in Types and Programming Languages. Foundations of Computing. MIT Press (2004)
28. Pohlers, W.: Proof Theory-The First Step into Impredicativity. Universitext, Spriger (2009)
29. Post, E.L.: Recursive unsolvability of a problem of thue. J. Symbolic Logic **12**, 1–11 (1947)
30. Prawitz, D.: Natural Deduction—A Proof-Theoretical Study. Dover (2006). Unabridged Republication. Original publication in Stockholm Studies in Philosophy series (1965)
31. Rédei, M. (ed.): John von Neumann: selected letters, volume 27 of History of Mathematics. American Mathematical Society/London Mathematical Society (2005)
32. Schwichtenberg, H., Wainer, S.S.: Proofs and Computations. ASL Perspectives in Logic, Cambridge University Press (2012)
33. The Univalent Foundations Program. Homotopy Type Theory: Univalent Foundations of Mathematics. Institute for Advanced Study (2013). https://www.homotopytypetheory.org/book
34. Troelstra, A.S., Schwichtenberg, H.: Basic proof theory. Number 43 in Cambridge Tracts in Theoretical Computer Science, 2nd edn. Cambridge University Press (2000)
35. Tseiten, G.S.: Associative calculus with unsolvable equivalence problem. Dokl. Akad. Nauk SSSR **107**(3), 370–371 (1956). In Russian
36. Turing, A.M.: On computable numbers, with an application to the entscheidungsproblem. Proc. London Math. Soc. (2) **42**, 230–265 (1936-7)

Index

M. Ayala-Rincón and F.L.C. de Moura, *Applied Logic for Computer Scientists*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-3-319-51653-0