

Copyright notice: These notes must not be distributed or copied in any form except for the purposes of attending or conducting the course CS209b at The University of Western Ontario.

Applied Logic for Computer Science

Notes for CS 209 b

H. Jürgensen

**Department of Computer Science
The University of Western Ontario
London, Ontario, 2006**

Copyright © 2006 by Helmut Jürgensen

1. Preface

The course *Applied Logic for Computer Science* was introduced as a mandatory course in the Computer Science curriculum at The University of Western Ontario in 1992. The present notes used in 2005/06 are a revised and updated version of my handwritten notes of 1992/93 and subsequent printed notes. They are being modified as the course goes on, and they will, most likely, again be modified in the next rounds of the course.

These notes are the only written material (beyond assignments and tests, including old tests) made available to the students. In particular, there is no prescribed textbook to be used by the students. Of course, the students are encouraged to look into other sources in the library and a few pointers to references are given in Chapter 2 of these notes. The students will base their work on the following items:

- (1) the lectures and their own notes from the lectures;
- (2) these course notes, which are available as PostScript files on the web;
- (3) exercises worked through and information provided in laboratory hours (these are scheduled as required and the material covered in them is not explicitly contained in these notes);
- (4) weekly assignments on material covered at the respective time in class.

Class attendance is mandatory and will be monitored by short randomly timed in-class tests. Performance is also monitored by possibly a midterm, a final exam or weekly exams. Examinations are open-book. The examination questions will be such that students working steadily, independently, diligently, and successfully on the assignments should have no problems passing the examinations — and the course.

The results of the earlier rounds of this course since 1992/93 have shown that there is a very strong correlation between course participation and independent work on assignments on the one hand and the outcomes of the examinations on the other hand.

The course notes are a very crude summary of the actual course. Relying on the notes instead of attending the classes is a very reliable step towards failing the course. The notes just summarize the material presented in class.

2. References

The following list of references is by no means a complete list of material relevant to the course. I only list sources used in preparing the notes and, in addition, provide a list of the names of a few specialized journals.

2.1. References — Books

- [ABR] S. Abramsky, D. M. Gabbay, T. S. E. Maibaum (editors): *Handbook of Logic For Computer Science*. Clarendon Press, Oxford, 1992.
Volume 1: Background: Mathematical Structures. Volume 2: Background: Computational Structures.

This two-volume set contains surveys and research papers on fundamental issues in logic. This is not a textbook, but a source of up-to-date information for researchers.

- [Bu1] S. N. Burris: *Logic for Mathematics and Computer Science*. Prentice Hall, Upper Saddle River, 1998.

An interesting complement to this course, taking a quite different approach. Focussing on classical logic with rather few applications relevant to computer science.

- [GR] G. Grätzer: *Universal Algebra*. D. van Nostrand, Princeton, 1968.

This is a classical book on universal algebra. The algebraic background on Herbrand universes can be found here.

- [Hinman] P. Hinman: *Fundamentals of Mathematical Logic*. Peters, Wellesley, 2005.

Quite different from this course, but useful reading.

- [LOR] P. Lorenzen: *Formale Logik*. Sammlung Götschen, Band 1176/1176a, Walter de Gruyter & Co., Berlin, 1958.

This is an older textbook, one of my favourites. Its approach and notation are, however, different from what is used in this course.

- [MCN] D. McNeill, P. Freiberger: *Fuzzy Logic. The Discovery of a Revolutionary Computer Technology—and how It Is Changing Our World*. Simon & Schuster, New York, 1993.

This is not a scientific book, but a reasonable introduction to the history and application of fuzzy sets and fuzzy logic. The book is written with enthusiasm about the subject and the people.

Sometimes I don't agree with the assessment of the historical importance of various developments and their originators. Nevertheless, the book is useful and easy to read.

[MONT] R. Montague: *Formal Philosophy. Selected Papers of Richard Montague*. Edited and with an Introduction by R. H. Thomason. Yale University Press, New Haven, 1974.

This collection of papers by Montague includes his famous work on universal grammar.

[NERODE] A. Nerode, R. A. Shore: *Logic for Applications*. Springer-Verlag, Berlin, 1993.

A very nice presentation of logic with applications in Computer Science. The approach taken is different from the one of this course.

[ROTH] C. H. Roth, Jr.: *Fundamentals of Logic Design*. Second Edition, West Publishing Company, St. Paul, 1979.

This is a typical textbook on circuit design. It is very useful as an introductory text, but does not cover advanced topics like races and hazards sufficiently.

[ROZ1] G. Rozenberg, A. Salomaa: *Cornerstones of Undecidability*. Manuscript, 1993.

Many fundamental questions of logic and computer science are discussed. This manuscript will be published as a book.

[RUB1] J. E. Rubin: *Mathematical Logic: Applications and Theory*. Saunders College Publishing, Philadelphia, 1990.

This is a typical undergraduate textbook. It contains lots of examples — in particular those about the Tufa island. Its approach and notation are different from the ones used in this course.

[SMU1] R. Smullyan: *What is the Name of this Book? The Riddle of Dracula and Other Logical Puzzles*.

This is a collection of logical puzzles in entertaining disguises followed by solutions and explanations. A main theme is that of the liar's paradox and self-reference (as in the title of the book). This is the source of the exercises involving the Portias, given in this course.

[SMU2] R. Smullyan: *Satan, Cantor, and Infinity, and Other Mind-Boggling Puzzles*. Alfred A. Knopf, New York, 1992.

This is a collection of logical puzzles like [SMU1]. The book contains some nice and small formal systems.

[SMU3] R. Smullyan: *To Mock a Mockingbird and Other Logic Puzzles, Including an Amazing Adventure in Combinatory Logic*. Alfred A. Knopf, New York, 1985.

From the cover: *In the first part of the book, he (the author) transports us once again to that wonderful realm where knights, knaves, twin sisters, quadruplet brothers, gods, demons, and mortals either always tell the truth or always lie... Inspector Craig of Scotland Yard gets involved in a search for the Fountain of Youth on the Island of Knights and Knaves. In the second and larger section, we accompany the Inspector on a summer-long adventure into the field of combinatory logic...*

[SMU4] R. Smullyan: *Gödel's Incompleteness Theorems*. Oxford University Press, New York, 1992.

This book provides a thorough discussion of how incompleteness arises and of what it means in mathematics.

[ZAD1] L. Zadeh, J. Kacprzyk (editors): *Fuzzy Logic for the Management of Uncertainty*. John Wiley & Sons, New York, 1992.

This is a collection of research papers and a few survey papers. The latter include a paper by NEAPOLITAN on uncertain approximate inference and a paper by PAWLAK on rough sets. Several of the research papers discuss axiomatizations of fuzzy logic and applications of fuzzy logic.

[ZHO1] L. Zhongwan: *Mathematical Logic for Computer Science*. World Scientific, Singapore, 1989.

This is the main basis of the course as far as mathematical logic itself is concerned. The title of the book is misleading. There is practically no computer science, nor any computer science application in it. The presentation is elegant and very concise.

[ZIM1] H. J. Zimmermann, L. A. Zadeh, B. R. Gaines (eds.): *Fuzzy Sets and Decision Analysis*. North-Holland, Amsterdam, 1984.

This is a collection of research papers, some reporting on applications of fuzzy set theory.

2.2. References — Specialized Journals

I list here some journals which routinely publish papers on logic including fuzzy logic. This list is by no means complete. Moreover, there are quite a few learned journals not devoted to logic in which articles related to logic can be found occasionally.

[AML] *Archiv für mathematische Logik und Grundlagenforschung*.

[JSL] *Journal of Symbolic Logic*

[FSS] *Fuzzy Sets and Systems*

[IJMMS] *International Journal of Man-Machine Studies*.

[ZMLGM] *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*.

[PHILS] *Philosophical Studies*

[JPHIL] *Journal of Philosophical Logic*

[AJAPS] *Annals of the Japanese Association for Philosophical Science*

[APAL] *Annals of Pure and Applied Logic*

[AIL] Алгебра и Логика (translated as *Algebra and Logic*)

[NDJL] *Notre Dame Journal of Formal Logic*

[AML] *Archive for Mathematical Logic*

[MIND] *Mind*

3. Logic and Computer Science

Formal logic has many applications in Computer Science, for example:

- basic proof techniques used in all parts of Computer Science (as in mathematics and the other sciences);
- problem analysis and software design (completeness, simplicity; for example, case analysis for tax report program);
- hardware specification (logic gates, switching theory);
- software specification (semantics of programming languages, program verification, correctness; for example, airport control system);
- process analysis, parallelism (specification, verification);
- automatic reasoning (expert systems, inference);
- logic programming
- imprecise reasoning (fuzzy logic, for example).

The course will address only some of these issues. However, the material is selected with these applications in mind.

Logic concerns the formal rules by which one reasons:

- the form of the statements is considered, not their meanings.

Example 3.1

When it rains the road is wet.	}	(premisses)
It rains.		
THEREFORE:		
The road is wet.		(conclusion)

In this example, it seems one uses the meaning of the statements.

Example 3.2

Whenever a bell rings a miracle happens.	}	(premisses)
A bell rings.		
THEREFORE:		
A miracle happens.		(conclusion)

Example 3.3

If y is a positive integer and a multiple of 3 then the sum of the digits of y is a multiple of 3.	}	(premisses)
x is a positive integer and a multiple of 3.		
THEREFORE:		
The sum of the digits of x is a multiple of 3.		(conclusion)

Example 3.4

The interior of every green UFO is full of flies from Mars.	}	(premisses)
Yesterday, a green UFO was seen in Victoria Park.		
THEREFORE:		
The interior of the UFO seen yesterday in Victoria Park was full of flies from Mars.		(conclusion)

These conclusions are valid once the premisses are accepted. In each case, the schema is as follows:

if p then q	}	(premisses)
p		
THEREFORE:		
q		(conclusion)

regardless of the meanings of p and q . To *express* such schemata we introduce a language:

- *Propositional Calculus*.

To *reason* about such schemes we need another language, a *meta-language*:

- English plus some shorthand notation.

Caution: Keep the language levels apart!

4. Basic Notions and Notation

We assume basic knowledge about sets, mappings, and relations. Some of the following items are just reminders or serve to fix notation in cases when different types of notation might have been used.

- \emptyset denotes the empty set. Not to be confused with the Greek letters φ and Φ .
- For a set A , $|A|$ denotes the cardinality of A . When A is finite, $|A|$ is the number of elements of A . In particular, $|\emptyset| = 0$.
- For sets A and B , $A \setminus B$ is the set of all those elements in A which are not elements of B .
- For sets A and B , $A \cap B$ and $A \cup B$ are the intersection and union of A and B , respectively.
- For sets A and B , A^B is the set of all mappings of B into A . In the special case of $B = \emptyset$, the set A^B consists of exactly one element.
- For a set A and a non-negative integer n , A^n is the set of all n -tuples of elements of A , that is,

$$A^n = \{(a_1, \dots, a_n) \mid a_1, \dots, a_n \in A\}.$$

In particular, when $n = 1$ then A^n can be identified with A and, when $n = 0$ then A^n is a singleton set consisting only of the empty tuple $()$.

- A set A is a subset of a set B , $A \subseteq B$, if and only if every element of A is an element of B .
- For two sets A and B , $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$.
- For two sets A and B , $A \subsetneq B$ if and only if $A \subseteq B$ and $A \neq B$. Distinguish this from $A \not\subseteq B$ which means that there is an element in A which is not an element of B .
- For a set A , 2^A denotes the set of all subsets of A . For example, if $A = \{1, 2, 3\}$ then

$$2^A = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \\ \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

Note that $2^\emptyset = \{\emptyset\}$. Hence $|2^\emptyset| = 1$ and, when A is finite, then $|2^A| = 2^{|A|}$.

We assume familiarity with proofs by induction of various kinds (by integers, by ordered sets, by structure).

- We use the words *function* and *mapping* interchangeably.
- \mathbb{N} is the set of positive integers and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. The cardinality of \mathbb{N} is denoted by \aleph_0 (aleph-zero).
- \mathbb{Z} is the set of integers.
- \mathbb{R} is the set of real numbers.
- An *alphabet* is a non-empty set. The elements of an alphabet are called *symbols* (or *letters*).

As a notational convenience, when one writes down *symbols*, one often uses letters (in the usual sense), possibly with a subscript, a superscript, or some other symbol (in the usual sense) like a bar, a hat etc. Thus, of course depending on the context,

$$a, a_{12}, a^{(2)}, \bar{a}, a', \hat{a}'$$

are examples of representations of symbols (in the mathematical sense). The “-” in \bar{a} , the “'” in a' and in \hat{a}' , and the “^” in $a^{(2)}$ have no special meanings. There is no *a priori* relation between the symbol a and any of the other symbols listed.

Let X be an alphabet.

- X^* is the set of all *strings* (or *words*) over X .
- λ denotes the *empty string* (or *empty word*).
- $X^+ = X^* \setminus \{\lambda\}$.

Words are *concatenated* to form new words. X^* and X^+ are closed under concatenation.

- A *language over X* is a subset of X^* .

Example 4.1 Let $X = \{(,)\}$ and let L be the set of all concatenations of nested pairs of brackets.

Some words in L :

$$(), \quad ((())), \quad ()(), \\ ((())())(), \quad (((())()), \quad ()()((()))$$

and some words not in L :

$$), \quad ((), \quad)($$

Note that this definition of L is not very precise; we need tools to specify languages in an unambiguous manner.

We shall introduce:

- the language of *propositional logic* and – later – the language of *predicate logic*;

To do so, we distinguish between:

- the *syntax* of the language and
- the *semantics* of the language.

Syntax concerns the arrangement of symbols; semantics concerns the meaning of words in the language.

Typically, the syntax of a language is defined using a *formal system*. The semantics is obtained via an *interpretation* of the words in some *world*.

In this course, we shall see several kinds of formal systems:

- *grammars*;
- *proof systems*;
- *abstract algebras*.

Formal systems have the following structure in common:

- an alphabet V ;
- a finite set $A \subseteq V^*$ of *axioms*;
- a finite set P of *rules* that determine how to create words from words;
- a *metarule* defining how rules are applied.

Grammars are used to specify languages.

Definition 4.1 A *context-free grammar* is a quadruple

$$G = (N, T, P, S)$$

where N is a finite alphabet (of *non-terminals*), T is a finite alphabet (of *terminals*) with $N \cap T = \emptyset$, P is a finite set of *rules* of the form $A \rightsquigarrow w$ with $A \in N$ and $w \in (N \cup T)^*$, and $S \in N$ is the *start symbol*.

To indicate the start symbol, we sometimes write G_B to denote a grammar derived from G , but with start symbol $B \in N$ instead of S .

Example 4.2 Let $N = \{S\}$, $T = \{(\,)\}$, and $P = \{S \rightsquigarrow SS, S \rightsquigarrow (S), S \rightsquigarrow \lambda\}$. The grammar to be considered is $G = (N, T, P, S)$.

Definition 4.2 Let G be a context-free grammar. Consider $w, v \in (N \cup T)^*$.

- (1) $w \approx > v$ (with respect to G) if and only if $w = w_1 A w_2$ for some $A \in N$ and $w_1, w_2 \in (N \cup T)^*$, $v = w_1 \bar{v} w_2$, and $A \rightsquigarrow \bar{v} \in P$.
- (2) $w \approx >^* v$ (with respect to G) if and only if there is an $n \in \mathbb{N}_0$ and there are words $w_0, \dots, w_n \in (N \cup T)^*$ such that $w = w_0$, $v = w_n$ and $w_i \approx > w_{i+1}$ for $i = 0, 1, \dots, n-1$.

The *language generated by G* is the set

$$L(G) = \{w \mid w \in T^*, S \approx >^* w\}.$$

Example 4.3 The grammar of Example 4.2 generates the language of Example 4.1. This language is called the *Dyck language D_1* .

Note that the symbols we are using may be non-standard; it is quite common in science that one adjusts the choice of symbols to the particular situation. For example, in formal language theory one normally uses the symbols \rightarrow and \Rightarrow instead of \rightsquigarrow and $\approx>$, respectively; I use the new symbols in the present context to avoid confusion with the logical symbols of \rightarrow for implication and \Rightarrow for logical consequence.

In a similar fashion some terms introduced in this course may be used in a different sense in another context or even in the logic literature. In combining several sources one should always verify that they use the same definitions for the same terms.

Students should realize that symbols are just that — symbols — and terms are just terms, that is, names. In particular, in the context of computer science, it should only be natural that one can distinguish between a symbol or a name and its various meanings, depending on the context.

Finally, in Table 4.1 we provide list of Greek letters typically used in mathematics together with their names. The Greek language has a few additional characters not commonly used in mathematics. We also give hints regarding the modern pronunciation of the letters in Greece; these hints should be taken with a very large grain of salt, that is, they should really just be considered as very simplistic approximations of reality. Typographical variants of these letters are often used as well.

álpha	α	A	run	nú	ν	N	run
béta	β	B	very	xí	ξ	Ξ	exam
gámma	γ	Γ	yard	ómikrón	\omicron	O	hobby
délta	δ	Δ	the	pí	π	Π	prompt
épsilon	ε	E	hell	rhó	ρ	P	rabbit
zéta	ζ	Z	Susan	sigma	σ	Σ	Susan
éta	η	H	sweet	táu	τ	T	tear
théta	ϑ	Θ	theme	úpsilon	υ	Υ	sweet
íota	ι	I	sweet	phí	φ	Φ	phone
káppa	κ	K	kitchen	chí	χ	X	Loch
lámabda	λ	Λ	Loch	psí	ψ	Ψ	topsy-turvy
mú	μ	M	home	óméga	ω	Ω	hobby

Table 4.1. The Greek letters typically used in mathematics: The first columns provide the names of the letters, the second and third columns give the lower case and upper case versions. The fourth columns roughly indicate the modern pronunciation. The accents indicate the correct stresses.

5. A Small Formal System

We present an example of a very small *formal system* to illustrate the notions of syntax and semantics. This system will be re-visited later in the course (it is adapted from [SMU4]).

- The alphabet V consists of five symbols,

$$V = \{\neg, p, n, (,)\}.$$

- We consider the language $L \subseteq V^*$ defined by

$$L = \bigcup_{w \in V^*} \{p(w), pn(w), \neg p(w), \neg pn(w)\}.$$

This language L could be defined using a context-free grammar (exercise!).

The words in L are exactly the syntactically correct ones in this example; thus L is a specification of the syntax.

To specify the semantics, we consider a machine which can print words in V^* .

- By the *norm* of a word w we mean the word $w(w)$.

The words in L are interpreted as statements about what the machine can or cannot print:

$$\begin{aligned} p(w) &\text{ means } w \text{ is printable;} \\ pn(w) &\text{ means the norm of } w \text{ is printable;} \\ \neg p(w) &\text{ means } w \text{ is not printable;} \\ \neg pn(w) &\text{ means the norm of } w \text{ is not printable;} \end{aligned}$$

We now assign truth values to words in L as follows:

$$\begin{aligned} p(w) &\text{ is true if and only if } w \text{ is printable;} \\ pn(w) &\text{ is true if and only if the norm of } w \text{ is printable;} \\ \neg p(w) &\text{ is true if and only if } w \text{ is not printable;} \\ \neg pn(w) &\text{ is true if and only if the norm of } w \text{ is not printable;} \end{aligned}$$

Now we know, for every word in L , what it means that the word is true or false.

We assume that the machine is perfect, that is, if it prints a word then the word is true.

- If it prints $p(w)$ then w is printable and the machine will also print w eventually.
- If it prints $pn(w)$ then $pn(w)$ is true and it will eventually print $w(w)$.
- If it prints $\neg p(w)$ then it will never print w .
- If it prints $\neg pn(w)$ then it will never print $w(w)$.

This machine describes its own properties.

- Consider the word

$$\neg pn(\neg pn).$$

Is it true? Is it printable? (Exercise!)

6. Propositional Calculus (Classical, Syntax)

We build the *language* \mathcal{L}^P of *propositional logic* using the following grammar:

$$\begin{aligned} N &= \{S, A, D, I\}, \\ T &= \{a, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (,), \neg, \wedge\}, \\ P &= \{S \rightsquigarrow A, S \rightsquigarrow (\neg S), S \rightsquigarrow (S \wedge S), \\ &\quad A \rightsquigarrow aD, \\ &\quad D \rightsquigarrow 1I, D \rightsquigarrow 2I, \dots, D \rightsquigarrow 9I, \\ &\quad D \rightsquigarrow 0, D \rightsquigarrow 1, D \rightsquigarrow 2, \dots, D \rightsquigarrow 9, \\ &\quad I \rightsquigarrow 0I, I \rightsquigarrow 1I, I \rightsquigarrow 2I, \dots, I \rightsquigarrow 9I, \\ &\quad I \rightsquigarrow 0, I \rightsquigarrow 1, I \rightsquigarrow 2, \dots, I \rightsquigarrow 9\}, \end{aligned}$$

and

$$G^P = (N, T, P, S).$$

Let $\mathcal{L}^P = L(G^P)$.

Let $G_A^P = (N, T, P, A)$ with N, T, P, A as above. The language $L(G_A^P)$ is called the set of *atoms* of \mathcal{L}^P . $L(G_A^P)$ is the set of words of the form $ax_1 \cdots x_n$ where $x_1 \cdots x_n$ is the decimal representation of a non-negative integer without leading zeroes. The elements in $\mathcal{L}_A^P = L(G_A^P)$ (the atoms) serve as an infinite set of names of variables.

The form of these names is not important — in the sequel we often use arbitrary symbols instead of these names (remembering that we could always have used these names instead). In particular, we often write $a_{x_1 \cdots x_n}$ instead of $ax_1 \cdots x_n$.

Here are some typical formulæ in \mathcal{L}^P :

$$\begin{aligned} &((\neg(a_1 \wedge a_{10})) \wedge (\neg a_2)) \\ &(\neg((\neg a_1) \wedge (\neg a_2))) \end{aligned}$$

Show how to get these from the grammar!

Theorem 6.1 *Let $f \in \mathcal{L}^P$. Then*

- *f is an atom of \mathcal{L}^P*

or

- *there is $f' \in \mathcal{L}^P$ such that $f = (\neg f')$*

or

- *there are $f_1, f_2 \in \mathcal{L}^P$ such that $f = (f_1 \wedge f_2)$.*

Proof: The atoms of \mathcal{L}^P are obtained when $S \rightsquigarrow A$ is applied as the first rule. Suppose, f is not an atom. There is a derivation

$$S \approx > w_1 \approx > w_2 \approx > \cdots \approx > w_n = f$$

with $w_1 \neq A$. Hence, $w_1 = (\neg S)$ or $w_1 = (S \wedge S)$. The occurrences of S in w_1 derive formulæ $f', f_1, f_2 \in \mathcal{L}^P$, respectively. Hence, f is of one of the two remaining forms. \square

Using Theorem 6.1, we are able to carry out proofs about formulæ and to define properties of formulæ by induction on their structure.

7. How to Find a Proof? (Heuristics I)

Finding a proof can sometimes be very difficult. This is inevitable. There are some techniques, however, which help to guide one in the process of finding a proof. They may not work in every instance, but trying them may be better than just sitting in front of the problem with no idea how to solve it. In this chapter, we prove a statement made in Section 6 and, at the same time, comment on the strategy used. To separate the comments which are not part of the proof from the proof itself, the comments are set in slanted letters and start and end with some stars. We prove the following statement:

Observation 7.1 *The language $L(G_A^p)$ is the set of all words of the form $ax_1 \cdots x_n$ where $x_1 \cdots x_n$ is the decimal representation of a non-negative integer without leading zeroes.*

Proof:

* * * * *

We first need to cast the statement into a form in which it is unambiguous what we have to prove. Therefore, we express items stated verbally in a rigorous formal way.

* * * * *

Let L denote the set of all words of the form $ax_1 \cdots x_n$ where $x_1 \cdots x_n$ is the decimal representation of a non-negative integer without leading zeroes. Then

$$L = \left\{ w \left| \begin{array}{l} w = ax_1x_2 \cdots x_n \text{ for some } n \in \mathbb{N} \text{ such that} \\ x_1 \in \{0, 1, 2, \dots, 9\} \text{ if } n = 1 \text{ and} \\ x_1 \in \{1, 2, \dots, 9\}, x_2, \dots, x_n \in \{0, 1, 2, \dots, 9\} \text{ if } n > 1 \end{array} \right. \right\}.$$

We have to prove $L(G_A^p) = L$.

* * * * *

Now separate the claim into manageable subclaims. When proving equality of two sets the usual approach is to prove mutual inclusion.

* * * * *

We prove this in two steps. First we show that $L \subseteq L(G_A^p)$. Then we show that $L(G_A^p) \subseteq L$.

* * * * *

Now we know what the major steps are, we attack them one by one. For each one we write out what the claim means and what we know.

* * * * *

To prove $L \subseteq L(G_A^p)$ we show that, if $w \in L$, then $w \in L(G_A^p)$. Therefore, let $w \in L$. Then there exist $n \in \mathbb{N}$ and $x_1, \dots, x_n \in T$ such that $w = ax_1 \cdots x_n$; moreover, $x_1 \in \{0, 1, 2, \dots, 9\}$ if $n = 1$; $x_1 \in \{1, 2, \dots, 9\}$ and $x_2, \dots, x_n \in \{0, 1, 2, \dots, 9\}$ if $n > 1$.

* * * * *

Now write out in detail what we have to prove.

* * * * *

We have to show that $w \in L(G_A^p)$, that is, $A \approx^* w$ and $w \in T^*$. The latter is true as $L \subseteq T^*$. For the former, we have to show that there exist $m \in \mathbb{N}_0$ and $w_0, w_1, \dots, w_m \in (N \cup T)^*$ such that

$$A = w_0 \approx w_1 \approx w_2 \approx \cdots w_{m-1} \approx w_m = w.$$

* * * * *

Up to this point, we have only expanded definitions. Now we have to find a derivation. This step may require some combination of stubborn work, ideas and trying out examples — fortunately not in our case. As w is described by cases, we now follow that case distinction (often a good idea).

* * * * *

We distinguish two cases:

Case $n = 1$: Then $w = ax_1$ with $x_1 \in \{0, 1, \dots, 9\}$. Then

$$A = w_0 \approx w_1 = aD \approx w_2 = ax_1 = w$$

is a derivation of w , where we used the rules $A \rightsquigarrow aD$ and $D \rightsquigarrow x_1$ in the first and second steps, respectively. Therefore, $w \in L(G_A^p)$.

Case $n > 1$: Then $w = ax_1x_2 \cdots x_n$ with $x_1 \in \{1, 2, \dots, 9\}$ and $x_2, \dots, x_n \in \{0, 1, \dots, 9\}$. In the first step we again have to apply the rule $A \rightsquigarrow aD$ as this is the only rule for A . Thus $w_1 = aD$. To get w_2 a rule for D has to be applied (as this is the only non-terminal in w_1). We apply the rule $D \rightsquigarrow x_1I$ to get $w_2 = ax_1I$.

For $i = 2, \dots, n - 1$ we derive w_{i+1} from w_i using the rule $I \rightsquigarrow x_iI$ with $x_i \in \{0, 1, \dots, 9\}$. In this way $w_{i+1} = ax_1 \cdots x_iI$. We show, by induction, that this is indeed a derivation.

First, let $i = 2$. If $n < 3$ nothing needs to be proved as $i > n - 1$. If $n \geq 3$ then $w_2 = ax_1I$ and, using the rule $I \rightsquigarrow x_2I$, one gets $w_3 = ax_1x_2I$.

Now assume the statement is true for $i < n - 1$. Thus $w_{i+1} = ax_1 \cdots x_iI$. We apply the rule $I \rightsquigarrow x_{i+1}I$ to get $w_{i+2} = ax_1 \cdots x_ix_{i+1}I$.

* * * * *

In this induction proof I have assumed that you are familiar with the basic principle of proof by induction. In this particularly simple case, a shortcut argument showing a few initial steps and then a generic step would have been acceptable.

* * * * *

Thus, we have constructed a derivation $A \approx_{>}^* w_n = ax_1 \cdots x_{n-1}I$. Now we apply the rule $I \rightsquigarrow x_n$ to get $w_n \approx_{>} w_{n+1} = ax_1 \cdots x_{n-1}x_n = w$ as required.

This proves that $w \in L(G_A^p)$ also when $n > 1$. This completes the proof of the inclusion $L \subseteq L(G_A^p)$.

* * * * *

Besides being rather detailed, the proof so far repeatedly uses certain constructs like $\{0, 1, \dots, 9\}$ or $\{1, 2, \dots, 9\}$. When this happens it is wise to consider introducing symbols for these constructs at the very beginning and then to use the symbols instead of the constructs. Thus one would re-write the proof accordingly. This is like using procedures in programming languages. When to do this is sometimes a matter of taste.

* * * * *

We now turn to the proof of the inclusion $L(G_A^p) \subseteq L$. We have to show that, if $w \in L(G_A^p)$, then $w \in L$.

* * * * *

Again we first state what we need to prove. Now we expand definitions again.

* * * * *

Let $w \in L(G_A^p)$. Then $w \in T^*$ and $A \approx_{>}^* w$, that is, there are $m \in \mathbb{N}_0$ and $w_0, w_1, \dots, w_m \in (N \cup T)^*$ such that

$$A = w_0 \approx_{>} w_1 \approx_{>} \cdots \approx_{>} w_m = w.$$

We have to show that there exist $n \in \mathbb{N}$ and $x_1, x_2, \dots, x_n \in T$ such that $w = ax_1 \cdots x_n$ and x_1, x_2, \dots, x_n satisfy the following conditions:

$$x_1 \in \begin{cases} \{0, 1, \dots, 9\}, & \text{if } n = 1, \\ \{1, 2, \dots, 9\}, & \text{if } n > 1, \end{cases}$$

and

$$x_2, \dots, x_n \in \{0, 1, \dots, 9\}.$$

* * * * *

The next part of this kind of proof for grammars and languages may become quite complicated as one has to conduct the proof for all possible derivations. Our special grammar is rather simple and, therefore, also the proof remains simple. We first try to say as much as possible about the derivation by looking at the rules of the grammar.

* * * * *

Now consider the derivation

$$A = w_0 \approx > w_1 \approx > \cdots \approx > w_m = w.$$

For A there is only the rule $A \rightsquigarrow aD$. Therefore, this rule must have been applied and $w_1 = aD$. Now we distinguish two cases.

Case $m = 2$: As $w_2 = w \in T^*$, the rule applied to w_1 must not introduce a non-terminal. Therefore, only a rule $D \rightsquigarrow x_1$ can have been applied with $x_1 \in \{0, 1, \dots, 9\}$. Thus $w = w_2 = ax_1$, hence $w \in L$.

Case $m > 2$: Then w_2 must contain a non-terminal; hence, only a rule $D \rightsquigarrow x_1I$ with $x_1 \in \{1, 2, \dots, 9\}$ can have been applied. Thus $w_2 = ax_1I$.

* * * * *

Now comes an induction proof showing that all further steps in the derivation, except the last one, have a specific property. Which property, depends on the grammar and may be sometimes less obvious than in this special case.

In our special case we observe that the only kind of rule we can keep applying without losing the non-terminal is of the form $I \rightsquigarrow xI$. This guides one in conjecturing what to prove.

In general, one looks for patterns in the derivation and tries to prove that these patterns are inevitable. The process is very similar to proving the correctness of a program: there, for every loop, one tries to formulate a condition that must be true upon entry of the loop, and then proves by induction that this is indeed so. That condition corresponds to the pattern mentioned before.

* * * * *

We show, by induction, that, for $2 \leq i < m$, one has $w_i = ax_1 \cdots x_{i-1}I$ for some $x_1 \in \{1, 2, \dots, 9\}$ and $x_2, \dots, x_{i-1} \in \{0, 1, \dots, 9\}$.

First, let $i = 2$. We have already shown that $w_2 = ax_1I$.

Now suppose that the statement is true for $i < m - 1$. We show that this implies that it is true for $i + 1$. By the assumption we have $w_i = ax_1 \cdots x_{i-1}I$. As $i + 1 < m$, w_{i+1} must contain a non-terminal symbol. The only rules applicable to w_i that introduce a non-terminal symbol are of the form $I \rightsquigarrow x_iI$ with $x_i \in \{0, 1, \dots, 9\}$. Therefore, such a rule must have been applied and $w_{i+1} = ax_1 \cdots x_{i-1}x_iI$.

* * * * *

The condition $i < m - 1$ is needed so that $i + 1$ does not exceed the permitted range, that is, $i + 1 < m$. On the other hand, we do not need to say anything about a lower bound on i . Clearly, $2 \leq i$ is needed. If $m \geq 4$ then $2 \leq i < m - 1$ is possible. If $m = 3$ the induction step is empty (and not needed).

* * * * *

Specifically, for $i = m - 1$, we get $w_{m-1} = ax_1 \cdots x_{m-2}I$ for some $x_1 \in \{1, 2, \dots, 9\}$ and $x_2, \dots, x_{m-2} \in \{0, 1, \dots, 9\}$. As $w_m \in T^*$, the rule applied to w_{m-1} to get w_m must not introduce a non-terminal. Therefore, it must be of the form $I \rightsquigarrow x_{m-1}$ for some $x_{m-1} \in \{0, 1, \dots, 9\}$. Thus $w = w_m = ax_1 \cdots x_{m-1}$ and $w \in L$ as required.

* * * * *

Note the use of the assumption $w_m \in T^$ and $w_i \notin T^*$ for $i < m$ throughout the proof. Be suspicious of a proof in which not all assumptions are used explicitly. Of course, it may happen that an assumption is really not needed. More frequently, however, this reveals that something is wrong with the proof. The usual way to detect which is the case is to expand every step in the proof into even greater detail. When you read a proof you should carefully watch for the way assumptions are used.*

* * * * *

This completes the proof of the inclusion $L(G_A^p) \subseteq L$. \square

A key part of the strategy used in this proof is to divide the problem into smaller subproblems, to subdivide these again, and so on, that is, solve the problem in small modules and then put these modules back together. This resembles very much the strategy of top-down design in program development.

Note the usage of the two induction proofs to establish subclaims. In this course it is assumed that you already know the *technique (structure)* of a proof by induction, that is, that you know what you need to do when asked to prove a statement by induction; of course, it may then still happen that you cannot actually do some parts of the proof, but that should not be a matter of the *structure* of the proof by induction.

At the end of this course, you should

- be able to distinguish a correct proof from an incorrect one (assuming they are of reasonably limited complexity),
- be able to find mistakes in (simple) incorrect arguments and isolate and correct them,
- be able to understand moderately complicated proofs,
- be able to find proofs of statements which are simple to prove,
- feel comfortable to try a proof even though, initially, you have no idea.

For these reasons, we shall cover a major part of (elementary) *proof theory* — the theory explaining what proofs are under various assumptions about the kind of logic¹ —; we shall also go through more proof heuristics as presented in this chapter.

It cannot be emphasized enough, however, that one cannot learn how to prove things without practising frequently — just as one cannot learn how to play a musical instrument without practising. Moreover, despite initial frustrations, in both cases, a certain level of proficiency can indeed be achieved if one does not give up right away.

¹ You will see that other kinds of logic than the one you are used to are reasonable and acceptable. You will also see that certain types of reasoning may be correct in one kind of logic and not in another.

8. Propositional Calculus (Classical, Semantics)

We now interpret the formulæ in \mathcal{L}^P as Boolean functions (Boole, 1815–1864). Let $\mathbb{B} = \{0, 1\}$. We interpret 0 as *false* and 1 as *true*. We shall frequently identify the Boolean values of 0 and 1 with the real numbers 0 and 1, respectively. This allows us to apply operations defined for real numbers to Boolean values.

Definition 8.1 Let $n \in \mathbb{N}_0$. A *Boolean function with n variables* is a mapping of \mathbb{B}^n into \mathbb{B} .

Definition 8.2 A (*Boolean*) *value assignment* is a function

$$v : \mathcal{L}_A^P \rightarrow \mathbb{B}.$$

Definition 8.3 Let v be a value assignment and $f \in \mathcal{L}^P$. The *value* $v(f)$ of f with respect to v is defined as follows:

- (1) If $f \in \mathcal{L}_A^P$ then $v(f)$ as defined above.
- (2) If $f = (\neg f')$ then $v(f) = 1 - v(f')$.
- (3) If $f = (f_1 \wedge f_2)$ then $v(f) = \min\{v(f_1), v(f_2)\}$.

By Theorem 6.1 this Definition 8.3 is complete. Obviously, $v(f)$ only depends on the values of those atoms that occur in f . Let $\alpha(f)$ be the set of atoms occurring in f . As usual, let $\mathbb{B}^{\alpha(f)}$ be the set of mappings of $\alpha(f)$ into \mathbb{B} , that is, the set of value assignments to the atoms occurring in f .

Now, when v varies over the atoms in $\alpha(f)$, then $v(f)$ varies accordingly. Thus, f defines a function

$$\beta(f) : \mathbb{B}^{\alpha(f)} \rightarrow \mathbb{B}$$

by

$$\beta(f)(v) = v(f)$$

for $v \in \mathbb{B}^{\alpha(f)}$. The function $\beta(f)$ is called the *Boolean interpretation of f* or the *Boolean function defined by f* .

Example 8.1 Let $f = (\neg a_1)$. Then $\alpha(f) = \{a_1\}$ and there are two mappings in $\mathbb{B}^{\alpha(f)}$, the one that maps a_1 onto 0 and the one that maps a_1 onto 1. Write these in form of a table, each row corresponding to one mapping. Call the first mapping v_0 and the second one v_1 (the names are arbitrary).

v	$v(a_1)$
v_0	0
v_1	1

Then $\beta(f)(v)$ is given by the following table:

v	$v(a_1)$	$\beta(f)(v)$
v_0	0	1
v_1	1	0

Example 8.2 Let $f = (a_1 \wedge a_2)$. Then $\alpha(f) = \{a_1, a_2\}$ and $\beta(f)$ is given by the following table.

v	$v(a_1)$	$v(a_2)$	$\beta(f)(v)$
v_0	0	0	0
v_1	0	1	0
v_2	1	0	0
v_3	1	1	1

In the sequel, we omit the column labelled v and all occurrences of v in the header line of such a table. A table like this is called a *truth table*.

Example 8.3 Let $f = ((\neg((\neg a_1) \wedge a_2)) \wedge a_3)$. To compute the truth table for the Boolean function $\beta(f)$ we organize the table as shown below. The computation starts by copying the values of the atoms into the table under the formula. We then compute “from the inside” writing the corresponding value under the symbol \neg or \wedge . The function value is found under the “outermost” such symbol (indicated by an arrow below).

a_1	a_2	a_3	$((\neg$	$((\neg$	$a_1)$	\wedge	$a_2))$	\wedge	$a_3)$
0	0	0	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	1	1
0	1	0	0	1	0	1	1	0	0
0	1	1	0	1	0	1	1	0	1
1	0	0	1	0	1	0	0	0	0
1	0	1	1	0	1	0	0	1	1
1	1	0	1	0	1	0	1	0	0
1	1	1	1	0	1	0	1	1	1

↑

Example 8.4 Let $f = (((\neg a_1) \wedge a_2) \wedge a_1)$.

a_1	a_2	$((\neg$	$a_1)$	\wedge	$a_2)$	\wedge	$a_1)$
0	0	1	0	0	0	0	0
0	1	1	0	1	1	0	0
1	0	0	1	0	0	0	1
1	1	0	1	0	1	0	1

↑

We consider all Boolean functions with 2 variables and find formulæ to express them. There are 16 Boolean functions with two variables as shown in Tables 8.1 and 8.2.

Of the symbols introduced in Tables 8.1 and 8.2, we use

$$0, 1, \vee, \rightarrow, \leftrightarrow$$

in the sequel to simplify formulæ. Moreover, in view of the Boolean interpretation, we omit brackets in many cases:

$$(a_1 \wedge (a_2 \wedge a_3)) \quad \text{and} \quad ((a_1 \wedge a_2) \wedge a_3)$$

define the same Boolean function². We use the following binding hierarchy:

$$\begin{array}{ll} \neg & \text{binds more strongly than } \wedge \\ \wedge & \text{binds more strongly than } \vee \\ \vee & \text{binds more strongly than } \rightarrow \\ \rightarrow & \text{binds more strongly than } \leftrightarrow \end{array}$$

Finally, we use arbitrary identifiers to denote atoms. Thus

$$\neg p \rightarrow p \wedge \neg q \vee r \leftrightarrow q$$

is equivalent to

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

The symbols $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$ are called *junctions* or *connectives*. We use big versions \bigwedge and \bigvee of \wedge and \vee in a similar way as one uses \prod and \sum for multiplication and addition.

In the sequel, let $\mathcal{L}^{\text{PROP}}$ be the set of formulæ involving arbitrary atoms, using the junctions $\neg, \wedge, \vee, \rightarrow$, and \leftrightarrow and following the binding rules introduced above. As an exercise, construct a context-free grammar generating that part of $\mathcal{L}^{\text{PROP}}$ which involves only atoms of \mathcal{L}^{P} . One observes the following analogue of Theorem 6.1: *If $f \in \mathcal{L}^{\text{PROP}}$ then*

- *f is an atom of $\mathcal{L}^{\text{PROP}}$*

or

- *$f = \neg f'$ for some $f' \in \mathcal{L}^{\text{PROP}}$*

² The mathematical details of this step are not dealt with in this course.

a_1	a_2	$F(a_1, a_2)$	$(a_1 \wedge (\neg a_1))$	constant function “0”
0	0	0		
0	1	0		
1	0	0		
1	1	0		
a_1	a_2	$F(a_1, a_2)$	$(a_1 \wedge a_2)$	conjunction
0	0	0		
0	1	0		
1	0	0		
1	1	1		
a_1	a_2	$F(a_1, a_2)$	$(a_1 \wedge (\neg a_2))$	
0	0	0		
0	1	0		
1	0	1		
1	1	0		
a_1	a_2	$F(a_1, a_2)$	a_1	
0	0	0		
0	1	0		
1	0	1		
1	1	1		
a_1	a_2	$F(a_1, a_2)$	$(\neg a_1) \wedge a_2$	
0	0	0		
0	1	1		
1	0	0		
1	1	0		
a_1	a_2	$F(a_1, a_2)$	a_2	
0	0	0		
0	1	1		
1	0	0		
1	1	1		
a_1	a_2	$F(a_1, a_2)$	$(\neg((\neg((\neg a_1) \wedge a_2)) \wedge (\neg(a_1 \wedge (\neg a_2)))))$	exclusive or, XOR, \oplus
0	0	0		
0	1	1		
1	0	1		
1	1	0		
a_1	a_2	$F(a_1, a_2)$	$\neg((\neg a_1) \wedge (\neg a_2))$	disjunction, OR, \vee
0	0	0		
0	1	1		
1	0	1		
1	1	1		

Table 8.1. 8 of the 16 Boolean functions with 2 variables.

a_1	a_2	$F(a_1, a_2)$	$((\neg a_1) \wedge (\neg a_2))$	NOR
0	0	1		
0	1	0		
1	0	0		
1	1	0		
a_1	a_2	$F(a_1, a_2)$	$(\neg((\neg((\neg a_1) \wedge (\neg a_2))) \wedge (\neg(a_1 \wedge a_2))))$	equivalence, \leftrightarrow
0	0	1		
0	1	0		
1	0	0		
1	1	1		
a_1	a_2	$F(a_1, a_2)$	$(\neg a_2)$	
0	0	1		
0	1	0		
1	0	1		
1	1	0		
a_1	a_2	$F(a_1, a_2)$	$(\neg((\neg a_1) \wedge a_2))$	implication, $a_2 \rightarrow a_1$
0	0	1		
0	1	0		
1	0	1		
1	1	1		
a_1	a_2	$F(a_1, a_2)$	$(\neg a_1)$	
0	0	1		
0	1	1		
1	0	0		
1	1	0		
a_1	a_2	$F(a_1, a_2)$	$(\neg(a_1 \wedge (\neg a_2)))$	implication $a_1 \rightarrow a_2$
0	0	1		
0	1	1		
1	0	0		
1	1	1		
a_1	a_2	$F(a_1, a_2)$	$(\neg(a_1 \wedge a_2))$	NAND
0	0	1		
0	1	1		
1	0	1		
1	1	0		
a_1	a_2	$F(a_1, a_2)$	$(\neg(a_1 \wedge (\neg a_1)))$	constant function "1"
0	0	1		
0	1	1		
1	0	1		
1	1	1		

Table 8.2. 8 of the 16 Boolean functions with 2 variables.

or there are $f_1, f_2 \in \mathcal{L}^{\text{PROP}}$ such that

- $f = f_1 \wedge f_2$

or

- $f = f_1 \vee f_2$

or

- $f = f_1 \rightarrow f_2$

or

- $f = f_1 \leftrightarrow f_2$

with parentheses used as required. For a model of how to build such a grammar look at the syntax for arithmetic expressions of a programming language like PASCAL.

Definition 8.4 A *literal* is an atom or the negation of an atom. Let $n \in \mathbb{N}$ and let $X = \{a_1, \dots, a_n\}$ be a set of distinct (names of) atoms. A *minterm* over X is a formula of the form

$$x_1 \wedge x_2 \wedge \dots \wedge x_n$$

where, for $i = 1, 2, \dots, n$, each x_i is either a_i or $\neg a_i$.

Theorem 8.1 Let $n \in \mathbb{N}_0$. There are $2^{(2^n)}$ Boolean functions with n variables. For $n > 0$, each of these is the interpretation of a formula in \mathcal{L}^{P} involving n atoms.

Proof: For n variables, there are 2^n possible assignments of Boolean values, corresponding to the 2^n Boolean n -tuples in \mathbb{B}^n .

A Boolean function with n variables is completely specified by providing its value for each of the 2^n value assignments to the variables. Hence every Boolean function can be specified by a Boolean 2^n -tuple. There are $2^{(2^n)}$ distinct Boolean 2^n -tuples. Hence this is the number of Boolean functions with n variables. This proves the first part of the claim.

For the second part³, consider a Boolean function F with n variables, a_1, \dots, a_n say where $n > 0$. We have to build a formula f such that $F = \beta(f)$. For all value assignments

$$v = (v_1, \dots, v_n) \in \mathbb{B}^n$$

such that $F(v) = 1$, construct the formula

$$f_v = (\hat{a}_1 \wedge \hat{a}_2 \wedge \dots \wedge \hat{a}_n)$$

where

$$\hat{a}_i = \begin{cases} a_i, & \text{if } v_i = 1, \\ (\neg a_i), & \text{if } v_i = 0, \end{cases}$$

³ The case of $n = 0$ is different. There are 2 Boolean functions with 0 variables, the constant functions 0 and 1. As the Boolean constants are not part of \mathcal{L}^{P} , we cannot describe them without using atoms. However, we get them using just 1 atom, that is, $a_1 \wedge \neg a_1$ and $a_1 \vee \neg a_1$ for 0 and 1, respectively.

for $i = 1, 2, \dots, n$. Note that each \hat{a}_i is a literal and that f_v is a minterm over the set $\{a_1, \dots, a_n\}$. Define

$$f = \begin{cases} \bigvee_{v \text{ with } F(v)=1} f_v, & \text{if such } v \text{ exist,} \\ a_1 \wedge \neg a_1, & \text{if no } v \text{ with } F(v) = 1 \text{ exists,} \end{cases}$$

that is, as the disjunction of the minterms that have been constructed if any or as a formula that always has the value 0 otherwise. We need to prove that for all $v = (v(a_1), \dots, v(a_n))$ one has $\beta(f)(v) = F(v)$.

Suppose, $F(v) = 1$. Then f_v occurs in the disjunction defining f and $f_v(v) = 1$. Thus $\beta(f)(v) = 1$.

Conversely, suppose $\beta(f)(v) = 1$. Then there is a minterm $f_{\bar{v}}$ in the disjunction defining f such that $f_{\bar{v}}(v) = 1$. We show that $\bar{v} = v$. Assume the contrary. Then there is i such that $\bar{v}(a_i) \neq v(a_i)$. Thus, if $v(a_i) = 1$ then $f_{\bar{v}}$ contains $(\neg a_i)$ in the conjunction and, if $v(a_i) = 0$ then $f_{\bar{v}}$ contains a_i . But then $f_{\bar{v}}(v) = 0$, a contradiction. Therefore, $\bar{v} = v$. Thus, f_v occurs in the disjunction defining f and, hence, by the definition of f , $F(v) = 1$.

This proves that $F(v) = 1$ if and only if $\beta(f)(v) = 1$, hence $F = \beta(f)$. \square

Example 8.5 Let F be the Boolean function with 3 variables as given by the following table:

a_1	0	0	0	0	1	1	1	1
a_2	0	0	1	1	0	0	1	1
a_3	0	1	0	1	0	1	0	1
$F(a_1, a_2, a_3)$	0	1	0	0	0	1	0	1

The value assignments for which $F(a_1, a_2, a_3) = 1$ are

$$v_1 = (0, 0, 1),$$

$$v_2 = (1, 0, 1),$$

and

$$v_3 = (1, 1, 1).$$

The corresponding minterms are

$$f_1 = (\neg a_1 \wedge \neg a_2 \wedge a_3),$$

$$f_2 = (a_1 \wedge \neg a_2 \wedge a_3),$$

and

$$f_3 = (a_1 \wedge a_2 \wedge a_3).$$

Thus f is given by the formula

$$\begin{aligned} f &= f_1 \vee f_2 \vee f_3 \\ &= (\neg a_1 \wedge \neg a_2 \wedge a_3) \vee (a_1 \wedge \neg a_2 \wedge a_3) \\ &\quad \vee (a_1 \wedge a_2 \wedge a_3). \end{aligned}$$

Let $\Sigma \subseteq \mathcal{L}^{\text{PROP}}$, and let v be a value assignment. The value $v(\Sigma)$ is given by

$$v(\Sigma) = \begin{cases} 1, & \text{if } v(f) = 1 \text{ for all } f \in \Sigma, \\ 0, & \text{otherwise.} \end{cases}$$

Definition 8.5 Let $\Sigma \subseteq \mathcal{L}^{\text{PROP}}$. The set Σ is said to be *satisfiable* if there is a value assignment v such that $v(\Sigma) = 1$. If v is a value assignment such that $v(\Sigma) = 1$ then we say that v *satisfies* Σ or that v is a *model of* Σ , and we write $v \models \Sigma$.

Definition 8.6 A formula $f \in \mathcal{L}^{\text{PROP}}$ is a *tautology* if $v(f) = 1$ for every value assignment v .

Definition 8.7 A formula $f \in \mathcal{L}^{\text{PROP}}$ is a *contradiction* if $v(f) = 0$ for every value assignment v .

Lemma 8.1 Let $f \in \mathcal{L}^{\text{PROP}}$. Then f is a contradiction if and only if $\neg f$ is a tautology.

Proof: f is a contradiction if and only if $v(f) = 0$ for every value assignment v . Now $v(\neg f) = 1 - v(f)$. Thus $v(f) = 0$ for every v if and only if $v(\neg f) = 1$ for every v and this is true if and only if $\neg f$ is a tautology. \square

Example 8.6 $a_1 \vee \neg a_1$ is a tautology.

Example 8.7 $a_1 \wedge \neg a_1$ is a contradiction.

9. Translating English into Formulæ (Part I)

We consider again some of the statements and conclusions made in Section 3 and translate them into formulæ. To do so we identify assertions and assign atoms to them as their *names*. The reasoning will be correct if the conclusions hold true regardless of the meaning of these names.

Example 9.1

When it rains the road is wet.	}	(premisses)
It rains.		
THEREFORE:		
The road is wet.		(conclusion)

Let a_1 mean “it rains” and a_2 “the road is wet. Then we get the form

$a_1 \rightarrow a_2$	}	(premisses)
a_1		
THEREFORE:		
a_2		(conclusion)

Now consider the truth table of $a_1 \rightarrow a_2$ for the case when a_1 has the value 1. Since the formula $a_1 \rightarrow a_2$ has the value 1 by assumption, we find that a_2 must have the value 1.

Not always is the translation so simple. Very often, the English syntax obscures the logical structure.

Example 9.2

The interior of every green UFO is full of flies from Mars.	}	(premisses)
Yesterday, a green UFO was seen in Victoria Park.		
THEREFORE:		
The interior of the UFO seen yesterday in Victoria Park was full of flies from Mars.		(conclusion)

Here let a_1 mean “X is a green UFO” and let a_2 mean “X’s interior is full of flies from Mars.” The first premiss then states $a_1 \rightarrow a_2$. The second premiss states “the X seen yesterday in Victoria Park was a green UFO.” With a bit of liberty we translate this also as a_1 , and we get the same schema.

The problem arising in Example 9.2 will be resolved later when we have a richer logic language.

Note that the translation from English into logic formulæ is not always unambiguous. One has to understand the English statements very precisely in order to achieve satisfactory translations.

Such problems are common everywhere in science, when you try to express reality in rigorous terms. These problems are very common in Software Engineering where one hardly ever gets a precise specification of the task a program has to perform, but where failure of the program can lead to disaster.

In translating a verbal specification into a formal one (English to logic, say) one discovers ambiguities and mistakes in the verbal specification and one clarifies the meaning of it.

10. Tautological Consequence

Consider propositions (statements)

$$p_1, \dots, p_n, p.$$

We want to determine how one may conclude p from $\{p_1, \dots, p_n\}$.

Suppose these propositions are expressed in terms of formulæ

$$f_1, \dots, f_n, f \in \mathcal{L}^{\text{PROP}}.$$

We study the following relation between the set $\{f_1, \dots, f_n\}$ and f :

“If v is a value assignment such that $v(\{f_1, \dots, f_n\}) = 1$ then $v(f) = 1$.”

Definition 10.1 Let $\Sigma \subseteq \mathcal{L}^{\text{PROP}}$ and $f \in \mathcal{L}^{\text{PROP}}$. f is a *tautological consequence* of Σ if, for every value assignment v , $v \models \Sigma$ implies $v \models f$. In this case we write $\Sigma \models f$.

We write $\Sigma \not\models f$ to denote the fact that it is not true that $\Sigma \models f$, that is, that there is a value assignment v such that $v(\Sigma) = 1$ and $v(f) = 0$.

For $f, g \in \mathcal{L}^{\text{PROP}}$ we write $f \models\!\!\!\models g$ if and only if $f \models g$ and $g \models f$.

Example 10.1 Let $\Sigma = \{a_1 \rightarrow a_2, a_1\}$ and $f = a_2$.

a_1	a_2	$a_1 \rightarrow a_2$	a_1	Σ	f
0	0	1	0	0	0
0	1	1	0	0	1
1	0	0	1	0	0
1	1	1	1	1	1

The value assignment v in the last row is the only one such that $v(\Sigma) = 1$, that is, $v \models \Sigma$. For this v , $v(f) = 1$. Hence $\Sigma \models f$.

Example 10.2 Let $f, g \in \mathcal{L}^{\text{PROP}}$ and $\Sigma = \{f \rightarrow g, f\}$. The only value assignments v that have $v(\Sigma) = 1$ are such that $v(f \rightarrow g) = 1$ and $v(f) = 1$. Hence, $\Sigma \models g$ regardless of what f and g are.

Note that Example 10.2 represents the type of reasoning shown in the examples of Section 3.

Example 10.3 Let $\Sigma = \emptyset$, $f = a_1 \vee \neg a_1$. From the definition of $v(\Sigma)$ one has that $v(\emptyset) = 1$ for all value assignments v .

a_1	Σ	f
0	1	1
1	1	1

Thus $\emptyset \models a_1 \vee \neg a_1$.

Lemma 10.1 $f \in \mathcal{L}^{\text{PROP}}$ is a tautology if and only if $\emptyset \models f$.

Proof: Note that $v(\emptyset) = 1$ for every value assignment v . Hence, $\emptyset \models f$ if and only if $v(f) = 1$ for every value assignment v , that is, if and only if f is a tautology. \square

Example 10.4 Let $f = a_1 \wedge a_2$ and $g = a_2 \wedge a_1$. We prove that $f \models g$.

a_1	a_2	f	g
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Hence, $v \models f$ implies $v \models g$, and $v \models g$ implies $v \models f$, that is, $f \models g$ and $g \models f$.

Lemma 10.2 Let $f, g, h \in \mathcal{L}^{\text{PROP}}$. Then $f \wedge g \models g \wedge f$ and $(f \wedge g) \wedge h \models f \wedge (g \wedge h)$.

Proof: Exercise. \square

Lemma 10.2 says that \wedge is commutative and associative. The latter justifies in part why we may omit brackets. A similar statement holds true for \vee .

Example 10.5 Show that $\neg\neg f \models f$ for all $f \in \mathcal{L}^{\text{PROP}}$.

f	$\neg f$	$\neg\neg f$
0	1	0
1	0	1

Hence, $v \models f$ implies $v \models \neg\neg f$, and $v \models \neg\neg f$ implies $v \models f$.

Note that there are other “natural” semantics for $\mathcal{L}^{\text{PROP}}$ in which the statement of Example 10.5 does not hold. Such semantics, different from the *classical* one discussed so far, will be presented later in this course.

Theorem 10.1 Let $\Sigma = \{f_1, \dots, f_n\} \subseteq \mathcal{L}^{\text{PROP}}$ and $f \in \mathcal{L}^{\text{PROP}}$. Then

$$\Sigma \models f \quad \text{if and only if} \quad \emptyset \models f_1 \wedge \dots \wedge f_n \rightarrow f.$$

Intuitively, this says that f is a tautological consequence of $\{f_1, \dots, f_n\}$ if and only if $f_1 \wedge \dots \wedge f_n \rightarrow f$ is a tautology. Note that in Theorem 10.1 the set Σ is assumed to be finite.

Proof: First, suppose that $\Sigma \models f$. Let v be an arbitrary value assignment and note that $v(\Sigma) = v(f_1 \wedge \dots \wedge f_n)$. Thus, if $v(\Sigma) = 0$ then $v(f_1 \wedge \dots \wedge f_n \rightarrow f) = 1$. If $v(\Sigma) = 1$ then $v(f) = 1$ and, again, $v(f_1 \wedge \dots \wedge f_n \rightarrow f) = 1$. Therefore, $f_1 \wedge \dots \wedge f_n \rightarrow f$ is a tautology.

For the converse, suppose that $f_1 \wedge \dots \wedge f_n \rightarrow f$ is a tautology, that is, its value is 1 for every value assignment. Therefore, if $v(f_1 \wedge \dots \wedge f_n) = 1$ then also $v(f) = 1$, hence $\Sigma \models f$. \square

Lemma 10.3 *Let $f, f' \in \mathcal{L}^{\text{PROP}}$. Then $f \models f'$ if and only if $v(f) = v(f')$ for all value assignments v .*

Proof: On the set $\mathbb{B} = \{0, 1\}$ consider the ordering $0 < 1$. $f \models f'$ holds if and only if $f \models f'$ and $f' \models f$. The former holds if and only if, for all value assignments v , $v(f') = 1$ whenever $v(f) = 1$, that is, $v(f) \leq v(f')$. Similarly, the latter holds if and only if $v(f') \leq v(f)$. Thus, $f \models f'$ if and only if $v(f) \leq v(f') \leq v(f)$ for all value assignments v , that is, $v(f) = v(f')$ for all value assignments v . \square

Remark 10.1 *The relation \models is an equivalence relation on the set of formulae, that is, \models has the following properties:*

- (a) *Reflexivity: For all f one has $f \models f$.*
- (b) *Symmetry: For all f and g , if $f \models g$ then $g \models f$.*
- (b) *Transitivity: For all f, g , and h , if $f \models g$ and $g \models h$ then $f \models h$.*

Proof: This follows from Lemma 10.3 (exercise!). \square

Lemma 10.4 *Consider $f, f', g, g' \in \mathcal{L}^{\text{PROP}}$ such that $f \models f'$ and $g \models g'$. Then the following statements hold true:*

- (a) $\neg f \models \neg f'$.
- (b) $f \wedge g \models f' \wedge g'$.

Proof: For (a), observe that $f \models f'$ if and only if, for every value assignment v , one has $v(f) = v(f')$ by Lemma 10.3. Therefore, $v(\neg f) = 1 - v(f) = 1 - v(f') = v(\neg f')$ for all value assignments v , that is, $\neg f \models \neg f'$, again by Lemma 10.3.

For (b), note that $f \models f'$ and $g \models g'$ imply that $v(f) = v(f')$ and $v(g) = v(g')$ for all value assignments v by Lemma 10.3. Therefore,

$$v(f \wedge g) = \min\{v(f), v(g)\} = \min\{v(f'), v(g')\} = v(f' \wedge g'),$$

that is, $f \wedge g \models f' \wedge g'$ by Lemma 10.3. \square

A statement similar to that of Lemma 10.4 also holds for \vee , \rightarrow , and \leftrightarrow .

Theorem 10.2 *Let $f, g, h, h' \in \mathcal{L}^P$ and $f \models g$. If h' results from h through replacement of some occurrences of f by g , then $h' \models h$.*

Proof: We distinguish several cases. First, note that if no replacement takes place then $h' = h$ and $h \models h'$ trivially by Remark 10.1. Hence, in the sequel assume that at least one occurrence of f is replaced by g .

Assume that $h = f$. Then $h' = g$ as there is a replacement. By $f \models g$, we have $h' \models h$.

Now suppose that $h \neq f$. We proceed by induction on the form of h using Theorem 6.1.

- (a) Suppose h is an atom. Then a replacement of an occurrence of f in h is only possible if $f = h$, which has been dealt with already.
- (b) Suppose the statement holds true for $h_1 \in \mathcal{L}^P$ and consider $h = \neg h_1$. The case of $h = f$ has been dealt with already. Hence, assume that $h \neq f$. Therefore, the occurrences of f being replaced must be in h_1 , resulting in $h'_1 \in \mathcal{L}^P$. Then $h' = \neg h'_1$. By induction assumption, $h_1 \models h'_1$, and by Lemma 10.4 $h \models h'$.
- (c) Suppose the statement holds true for $h_1, h_2 \in \mathcal{L}^P$ and consider $h = h_1 \wedge h_2$. The case of $h = f$ has already been dealt with. Hence assume that $h \neq f$. Therefore, the occurrences of f being replaced are in h_1 or h_2 , resulting in h'_1 and h'_2 , respectively. Hence $h' = h'_1 \wedge h'_2$. By induction assumption, $h_1 \models h'_1$ and $h_2 \models h'_2$, and by Lemma 10.4, $h \models h'$.

□

Example 10.6 Let $f = a_2$, $g = a_2 \wedge a_2$, $h = a_1 \wedge a_2$. Replacing f in h by g results in $h' = a_1 \wedge a_2 \wedge a_2$. As $f \models g$, we have $h \models h'$.

In Example 10.6 one sees the rudiments of a calculus of propositions: One performs certain operations on formulæ without changing their truth values — and, ultimately, not using value assignments. This idea will be expanded later. As an exercise, prove the analogue of Theorem 10.2 for $\mathcal{L}^{\text{PROP}}$ instead of \mathcal{L}^P .

Theorem 10.3 *Let f be a formula using \neg , \wedge , and \vee only with all conjunctions and all disjunctions enclosed in parentheses. Consider the following operator ∂ on such formulæ:*

∂f results from f by exchanging \vee and \wedge and by replacing every atom by its negation.

Then $\partial f \models \neg f$.

Proof: If f is an atom then $\partial f = \neg f$ and the statement follows trivially by Remark 10.1.

If f is not an atom, then f is of one of the following forms⁴: $f = \neg f_1$ or $f = f_1 \wedge f_2$ or $f = f_1 \vee f_2$. Assume that f_1 and f_2 have the required property.

If $f = \neg f_1$ then $\partial f = \neg \partial f_1$. By the induction assumption, $\partial f_1 \models \neg f_1$. Therefore, $\partial f \models \neg f$.

⁴ We did not prove this statement which is analogous to Theorem 6.1. A formal proof can be achieved using Theorem 6.1 and the definition of \vee in Tables 8.1 and 8.2.

If $f = f_1 \wedge f_2$ then $\partial f = \partial f_1 \vee \partial f_2$. By the induction assumption, $\partial f_1 \models \neg f_1$ and $\partial f_2 \models \neg f_2$. Therefore, $\partial f \models \neg f_1 \vee \neg f_2$ and $\neg f_1 \vee \neg f_2 \models \neg(f_1 \wedge f_2)$, hence $\partial f \models \neg f$.

If $f = f_1 \vee f_2$ the $\partial f = \partial f_1 \wedge \partial f_2$. By the induction assumption, $\partial f_1 \models \neg f_1$ and $\partial f_2 \models \neg f_2$. Therefore, $\partial f \models \neg f_1 \wedge \neg f_2$ and $\neg f_1 \wedge \neg f_2 \models \neg(f_1 \vee f_2)$, hence $\partial f \models \neg f$. \square

The operation ∂ on formulæ is called the *duality operator*. Theorem 10.3 provides a formal method to find the negation of a formula⁵.

Example 10.7 Let $f = (a_1 \wedge \neg a_2) \vee (a_3 \wedge a_4)$. To negate this, apply Theorem 10.3 to find that $\neg f \models (\neg a_1 \vee \neg \neg a_2) \wedge (\neg a_3 \vee \neg a_4)$. Note that, in classical logic, this can be further simplified using Example 10.5.

Some results in this section are stated only for formulæ in \mathcal{L}^P , but also hold for formulæ involving the additional junctors \vee , \rightarrow , and \leftrightarrow . We chose to prove them for \mathcal{L}^P only because, in this case, we could use Theorem 6.1. These proofs carry over to the more general kind of formulæ, as far as the semantics is concerned, due to the fact that every Boolean function can be obtained as the interpretation of a formula in \mathcal{L}^P . This works except in those cases where explicit syntactic assumptions are made as in Theorem 10.3.

⁵ We use the symbol ∂ to denote the duality operator *in this course only*. Textbooks may use a different symbol or none at all, that is, you should not expect this meaning of the symbol ∂ to be commonly understood.